An Artificial Intelligence Knowledge-Based Fault Detection and Isolation System

By Itay Bar

Word Count: 9,611 excluding titles, tables of contents and appendixes.

BSc Information Systems and Management Project Report

Department of Computer Science and Information Systems,
Birkbeck College, University of London

May 2014.

This report is the result of my own work except where explicitly stated in the text.
The report may be freely copied and distributed provided the source is explicitly acknowledged.

1. Abstract

Detection of faults and errors in complex systems is a critical and challenging procedure. The increasing requirements on performance and reliability of such complex systems have created a need for advanced Fault Detection and Isolation (FDI) systems.

This project looks at FDI in the sector of distributed Electronic Point of Sale (EPOS) networks in large venues and focuses on Fortress, a global provider of customer management and stored value solutions for the Sports & Entertainment industry.

This report details the development of an Artificial Intelligence Fault Detection and Isolation (FDI) system using a Knowledge-Based (KB) System. The FDI solution employs rules from a rule-base to detect faults in a Fortress EPOS solution called TMC. When a fault is detected, its source is isolated and notifications are sent to the relevant Fortress support team.

2. Contents

3. Introduction

3.1 Background

Detection of faults and errors in complex systems is a critical and demanding operation. The problem of Fault Detection has received a lot of attention in fields of reliability engineering, control, and computer sciences. The increasing requirements on performance and reliability of such complex systems have created a necessity for smart, robust and sophisticated Fault Detection and Isolation (FDI) systems (Sampath et al, 1996, Isermann and Balle, 1997).

This project looks at the sector of distributed Electronic Point of Sale (EPOS) networks in large venues and focuses on Fortress, a global provider of customer management and stored value solutions for the Sports & Entertainment industry. One of the solutions Fortress provides is an Electronic Cash (RFID based E-Cash) Point-of-Sale system called ***TMC***. The TMC software runs on Windows machine with no screen, and has a very minimal UI built on-top of an RFID reader 8-Lines LCD display. Due to its limited interface, when faults happen to the system, they are only detected when used.

This report details the development of a Fault Detection and Isolation (FDI) system using a Knowledge-Based (KB) System. A monitoring service connects to each TMC machine on a fixed schedule and gets its current state, run a series of checks using the FDI System, which employs rules from a rule-base to detect faults in the system. When a fault is found the service will send a notification (via Email) the appropriate Fortress support team (i.e. hardware or software).

3.2 Aims and Objectives

3.2.1 Aims and Objectives

The aim of this project is to design and build a Fault Detection and Isolation (FDI) solution which routinely checks the state of a propriety software/hardware solution (called TMC or TMCX). The system uses a knowledge-based system, which employs rules to detect faults and send notification to specific support teams. The system should also provide a web-based explanation facility with a high-level overview of the current state of each TMC it monitors.

To achieve the goals of this project, the following sub-systems will be designed and implemented:

- A Knowledge-Based (KB) Expert System for Fault Detection and Isolation (FDI).
- A Monitoring service that periodically checks a collection of TMCs for faults using the FDI system.

- A Web API and a Web Based explanation facility for the end-users to review and understand the FDI system results.

3.2.2 Motivation

By implementing the system, Fortress will benefit from the ability to monitor the state of all the TMCs in all venues it supplies this solution to. The main goal is to detect faults, and resolve them in advance, and give Fortress's clients (and their customers) a much better experience.

Achieving this goal will also reduce pressure from the support teams on "event days" (e.g. when issues happen while customers are in queue to pay) and streamline the operations of both teams.

All the information collected by the system will be kept for later analysis, enabling Fortress to later build a Predictive Maintenance solution which could be used to predict faults before they happen.

3.3 Methodology

Due to the nature of this particular project a combination of methodologies were used for the development of the solution. These can be split into three categories:

1. The process of building Expert System is called "***Knowledge Engineering***" and it involves *Knowledge Acquisition* and *Knowledge Representation*. The knowledge acquisition methods used in this project were interviews and surveys with the TMC developers (domain expert). Code Review of the TMC was also done (with the developers) to better understand and discover possible faults (termed "***fault cases***").The Knowledge representation technique was to formalise, through an iterative process, the knowledge of the domain experts into tests (termed "***fault-case tests***"), comprised of sets of rules (termed "***fault-case tests rules***").
2. ***The Fault Detection and Isolation (FDI)*** approach taken in this project was a hybrid between the two most common FDI technique "Model-Based" and "Signal-Processing".
3. The ***software engineering*** methodology used was the Dynamic Software Development Method (DSDM), a structured agile development approach.

These are explained in greater detail in sections 5 and 6 of this report.

In order to achieve the proposed solution, the system was to be split into a several sub-systems:

- A Fault Detection and Isolation (FDI) knowledge-based (KB) expert system.
- A monitoring service that periodically runs the FDI checks on all the TMCs on a client site.
- A Web API that provides the current state of each TMC the service monitors.
- A Web-based explanation facility which will be used by the end-users.

The work on this project was divided into time-boxed stages representing different milestones:

- Stage 1 – Preparations and Knowledge Acquisition (November, 2013):
- Stage 2 – Specifications, Documentations and First Implementation (December, 2013):
- Stage 3 – Analysis, Further Development and Alpha Testing (January, 2014):
- Stage 4 – Verification, Validation and Beta Testing (March, 2014):
- Stage 5 – Report (April, 2014):

3.4 Report Structure

The report starts at section 4 with a review of Knowledge-Based (KB) Systems and, Fault Detection and Isolation (FDI). This section provides a brief history of Artificial Intelligence and Expert Systems, describes the process of designing Expert Systems and gives an overview of the different approaches of FDI.

Section 5 explains the different methodologies used in the development of this project including the Knowledge Engineering Methodologies, the FDI approach employed and the Software Engineering method used.

Section 6 details the development and implementation process and introduces the technologies used in the project. Additionally, the architecture, design choices, and all phases of the development process are explained in detail.

Section 7 describes the testing done throughout the development of this solution, including the functional and verification testing, the validation and operational testing and the beta (pre-production) testing to be done post-submission.

Section 8 presents the conclusions of the report and development process through reflection and summary and opens the discussion about further development.

Sections 9 and 10 provide references and appendixes respectively.

4. Review of Knowledge-Based (KB) Systems and, Fault Detection and Isolation (FDI)

4.1 History of Artificial Intelligence (AI) and Expert Systems

The dominating view of problem solving in early Artificial Intelligence (AI) research from the 1940's to 1960's was of a general-purpose algorithms trying to combine basic reasoning procedures to achieve complete solutions. This approach has been termed "**weak method**" because it employs weak knowledge about the domain of the problem being solved. This view produced unsatisfactory performances for complicated problems, in complex domains (Russell and Norvig, 1995).

One of the key findings in AI research was the understanding that the domain for intelligent machines has to be restricted and clearly defined (Negnevitsky, 2002). Restricting the domain enables the delivery of practical results by solving typical cases in narrow areas of expertise, allowing larger reasoning.

In 1965 a team of two computer scientists (Edward Feigenbaum and Bruce Buchanan) and a genetics Nobel Prize winner (Joshua Lederberg) was formed at Stanford University to build the *DENDRAL* project, a computer program to determine the molecular structure of Martian soil. Since there was no scientist algorithm for this procedure, the idea was to incorporate the expertise of Lederberg into a computer program, aimed to perform at a human expert level (Negnevitsky, 2002, Russell and Norvig, 1995).

The importance of DENREAL was that it was arguably the first successful knowledge-based system, based on domain-specific rules (Russell and Norvig, 1995). This represented a major paradigm shift in AI, from board general purpose weak methods to domain-specific, knowledge-intensive techniques. Feigenbaum and others at Stanford continued to investigate the possible applications of the new approach, termed "**expert systems**" and began developing *MYCIN*, a blood infections diagnosis program.

Using heuristics in the form of high-quality domain-specific rules, the *DENDRAL* and *MYCIN* teams demonstrated that computers could equal an expert in a narrow, well defined domain.

These projects introduced the concepts of Expert Systems, Knowledge Engineering and Knowledge Based (KB) Expert Systems, a field which continued to flourish both in research and commercial use since then and through the 1980's (Russell and Norvig, 1995).

4.2 Designing Expert Systems

An "expert system" is a program that achieves a high level of performance on problems that are of a difficulty level which requires human expertise to be solved (Feigenbaum, 1984). A knowledge-based system is software (or software packages) that can be thought intelligent in some narrow area of expertise (Negnevitsky, 2002). Knowledge-based Expert Systems are systems that emulate the decision making process and ability of a human expert (Giarratano and Riley, 1998) and are designed to help humans in scenarios where an expert might be needed.

In a basic expert system, there are two components (Figure 4.2.1) a ***knowledge-base*** which contains all the relevant expert knowledge and an ***inference engine*** which draws conclusions based on the knowledge from knowledge-base (Giarratano and Riley, 1998).

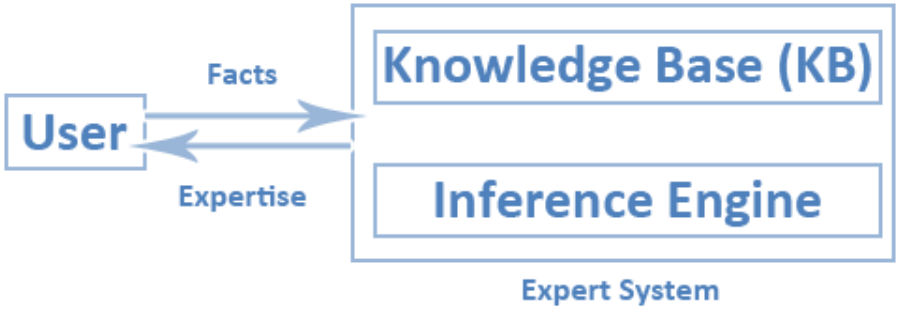


Figure 4.2.1 – Expert System Structure (based on Giarratano and Riley 1998)

4.2.1 Knowledge-Based Expert System Development Roles

A knowledge based development team usually consists of multiple members preforming five key roles, detailed in Table 4.2.1 below (Negnevitsky, 2002):

Role Name	Description / Responsibility
Domain Expert	A knowledgeable and skilled individual capable of solving problems in a specific domain.
Knowledge Engineer	In charge of acquiring knowledge from the domain expert, designing, building and testing the system with the help of the domain expert.
Programmer	In charge of actual programming and translating the domain knowledge to computer language, including the knowledge and data representation structures (knowledge-base), control structures (inference engine) and user interface.
Project Manager	Responsible of management tasks such as interacting with all team members, communicating issues and keeping the project on track to ensure deliverables and milestones are met.
End-User	The person(s) who would be using the system once developed. The system must satisfy the specific needs of the end-users and pass their acceptance tests.

Table 4.2.1 – Knowledge-Based Expert Systems Team Roles (Based on Negnevitsky, 2002)

Figure 4.2.2.1 details the process of building expert systems is called ***knowledge engineering***, which involves (Giarratano and Riley, 1998):

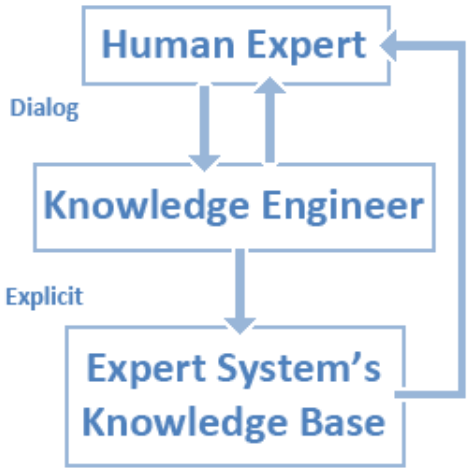


Figure 4.2.2.1 – The Process of Knowledge Engineering (based on Giarratano and Riley, 1998)

- ***Knowledge Acquisition*** – elicit the required knowledge from an *expert* or other sources using different methods such as interviewing, observing and researching about the particular domain.
- ***Knowledge Representation*** – after enough knowledge has been gathered, it needs to be explicitly codified into the system and represented in a way which the inference engine can use for its reasoning.

4.2.2.1 Knowledge Acquisition

Knowledge acquisition is the processes of collecting all the relevant knowledge, from experts and other sources. This process should happen before and during the process of codifying the knowledge into formal representation (Russel and Norvig, 1995).

There are different techniques for extracting and capturing knowledge, such as interviews, observations, questioners, etc. These processes are manual and require a lot of time. Feigenbaugm (1984) refers to it as the "bottleneck problem of AI", and emphasise another layer of the complexity, that is, it's not only enough to extract the relevant information from the experts, but it's also crucial how it is transferred into computer programs.

The techniques and methods for knowledge acquisition used in this project were interviews, surveys and code review. This process is discussed later in sections 5 and 6.

4.2.2.2 Knowledge representation in Rule-Based Systems

In a rule-based system, the knowledge is represented by a set of "IF-THEN" rules. Rules consist of two parts, "IF" (antecedent) which relates to given information or facts, and "THEN" (consequent) part which relates to the action to be taken. Basic examples rules are:

- **IF** "Printer" is "Off" **THEN** Action is *"Notify Support"*
- **IF** "Printer" is "Off" **OR** "Printer" is "Out Of Paper" **THEN** Action is "Notify Support"
- **IF** "Main Server" is "Disconnected**" AND** "Secondary Server" is "Disconnected" **THEN** Action is "Notify Support"

Rules specify a relation, recommendation, instruction, strategy or heuristic. When the condition (antecedent) part of a rule is satisfied, the rule is said to fire and the action part (consequent) is executed. Rules provide information about how to solve a problem and are efficient way to represent knowledge as most expert are capable of expressing their knowledge is such form (Negnevitsky, 2002).

The stage of codifying expert knowledge into rules is iterative, and based on the experts critique. Rules are created, examined and refined until the expert is satisfied with the systems performances (Giarratano and Riley, 1998).

The approach used for formalising rules in this project is discussed later in section 6.

4.2.2.3 Inference Methods

In a rule-based system the inference engine decides which rule antecedents were satisfied by the facts or input. There are two main methods used in problem-solving strategies (Giarratano and Riley, 1998):

- **Forward Chaining** – in this method, reasoning starts from facts and moves to the conclusion. For example, receipts are not printed (fact) then the printer has a fault (conclusion).
- **Backward Chaining** – in this method, reasoning is done in reverse, from a hypothesis (a probable conclusion), to the facts that support the hypothesis. For example the printer might not print (hypothesis), so we check if it is on, connected and has paper (facts) – if all are true, then the printer is not faulty. Hypothesis can be looked at as goals to be proven (or disproven).

An inference engine can use one of the above methods, the choice of which to use depends on the problem to be solved. Diagnostic (detection) problems are better solved with backwards chaining,

whereas prognosis (prediction) problems are more suitable for forward chaining (Giarratano and Riley, 1998, Negnevitsky, 2002).

Negnevitsky (2002) suggests that a way to choose which method to use is to study how the domain experts solve the problem, if the experts first need to gather facts, and then infer information from it, chose forward chaining. If the experts start with a hypothetical solution, then attempt to find facts that prove the proposed solution, chose backwards chaining.

The approach we used in this project is backward-chaining because it's more suitable for detection problems and matches the way the domain expert (TMC developers) solve problems. This is discussed later in section 6.5

4.3 The Structure of an Expert System

Figure 4.3.1 details a basic rule-based knowledge-based system:

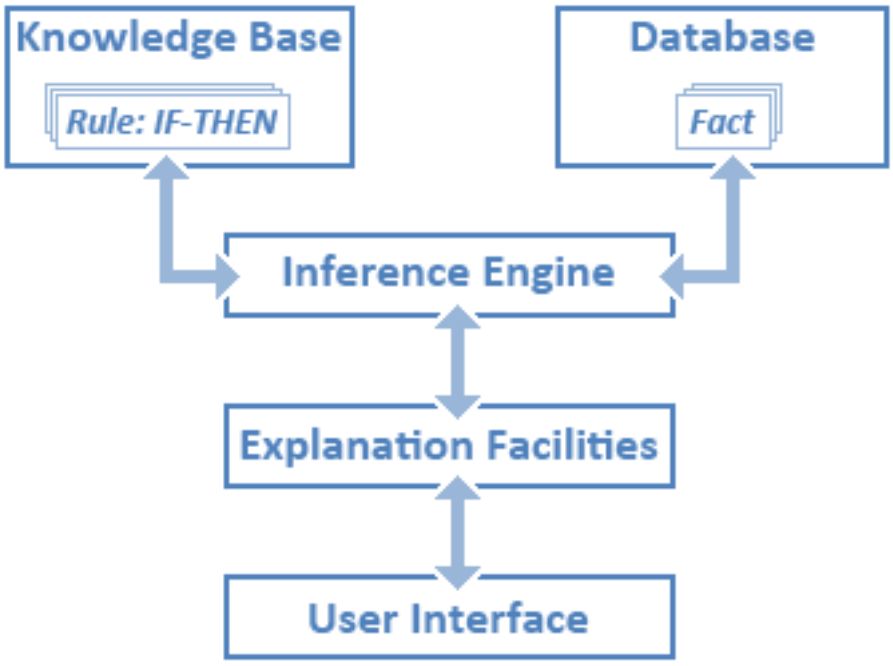


Figure 4.3.1 – KB Expert System Structure (based on Negnevitsky, 2002)

- **Knowledge-Base** – contains the domain knowledge (elicited from the domain expert) useful for problem solving. In a rule-based expert system, knowledge is represented by rules.
- **Database** – includes sets of facts and other information to be matched in the IF part of the rules.
- **Inference Engine** – carries out the reasoning process with the information from the database and rules from the knowledge-base.

- **Explanation Facilities** – Allows the user to see how and why a particular conclusion has been reached.
- **User Interface** – The UI is the means of communication between the end-user and the system.

4.5 Fault Detection and Isolation (FDI)

A fault detection system is an external (operator) system that is used to detect operational errors or issues and isolate their source and relevance in the system (Chen and Patton, 1999). Fault Detection and Isolation (FDI) techniques are classified into two main categories – Model-Based and Signal-Based processing.

4.5.1 Model-Based FDI

The concept of Model-Based FDI is to compare the state or behaviour of an actual system to that of a fault-free system (Sedighi et al, 2013). After gathering the state of the system, the fault detection process consists of two stages (Ding, 2008):

- **Residual generation** – comparing the state of the actual system to the fault-free state model and generating residuals – differences between the states. If after comparing the states, there are no residuals that means there are no faults.
- **Residual evaluation** – processing and analysing the residuals and, the decision making logic related to diagnosing faults and the sources.

4.5.2 Signal-Processing Based FDI

Signal-processing Based FDI work on the assumption that the systems signal contains information about its state and possible fault symptoms. This approach is more suited for systems which a) have such signals, and b) are in a steady state, and therefore is less efficient for dynamic systems with a large range of possible states (Ding, 2008).

The FDI approach taken in this project is a hybrid between Model-Based and Signal-Processing and is explained in section 5.

5. Methodology

This section explains the various Knowledge Engineering, Fault Detection and Isolation, and Software Engineering methods and methodologies used in this project.

5.1 Knowledge Engineering and Roles of the Development Team Members

Table 5.1.1 lists the key roles in the development team and who performed them in this project.

Role	Actor(s) in this project	Notes
Domain Expert	TMC Developers	The most knowledge-aware individuals in this domain are the TMC Developers. The developers of the TMC liked the idea of this project greatly and were very keen on helping and giving their knowledge.
Knowledge Engineer	Author	For this project, the author performed the role of the knowledge engineer and transferred the knowledge of the TMC developers into rules. This process is explained further in section 6.
Programmer	Author	All the projects deliverables, a part from pre-existing Fortress Web Services and Database Store Procedures were developed and written by the author.
Project Manager	Fortress Director of Software Development and Author.	The Fortress Director of SW Development had to approve this project and allocate time for the TMC Developer and Support Teams to work on it. Once the project was approved, the base instruction was that this project is to interfere as little as possible with the day-to-day work of the TMC developers, and as long as that was kept, the director of development did not interfere.
End-User	Fortress Support Teams (Software and Hardware)	The end users are the Fortress Support teams (both Software and Hardware), as they will be the ones notified when faults occur and will be in charge of resolving issues.

Table 5.1.1 – Development Team Member and Roles

5.1.1 Knowledge Engineering

The knowledge acquisition process was comprised of

- Interviewing and surveying the TMC developers (domain expert)
- Review the TMC code to gather the information needed to design the rule-based knowledge-base.
- Interview the support teams and collect feedback on different issues and faults usually detected in "real life" environment.

After the 1st iteration of interviews, a basic understanding of how the knowledge should be represented was formed. It was decided that a collection possible faults (termed "***fault cases***") would be created. Each fault case would have associated tests (termed "***fault case tests***") employing rules to determine if the fault is present or not. Table 5.1.1.1 details a basic fault case and its rules:

Fault Case Name/Description **(Suggested by the Domain Expert)**	TMC Offline
Fault Case Tests **(Defined by the Domain Expert)**	**1. Check if TMC Online** • Is the TMC connected to a DCS? • Is the TMC connected to an SDS? *"DCS"* and "*SDS*" are two different prosperity Fortress Servers the TMC can communicate with.
Resulting Rule(s) **(Translated by the Knowledge Engineer)**	***IF*** *(Comm.Comm_Data.Status.DCS_State is "true")* ***OR*** *(Comm.Comm_Data.Status.SDS_State is "true")* ***THEN*** *"Fault Not Present".*

Table 5.1.1.1– Example of a simple Fault Case

Through several iterations of the process described above, a collection of fault cases (and tests) was created. They were then translated into rules, which were reviewed by the domain expert. Once approved, they were saved in the knowledge-base.

There were no user inputs needed, and all the information (facts) is provided by the TMC Status String and the Execution Environment. This is explained in detail in section 6.

5.1.2 KB System Structure

The system structure for this project is slightly different from the basic knowledge-based expert system described in section 4. Figure 5.1.2.1 describes the system structure for this project and how it deviates from the basic structure, and why.

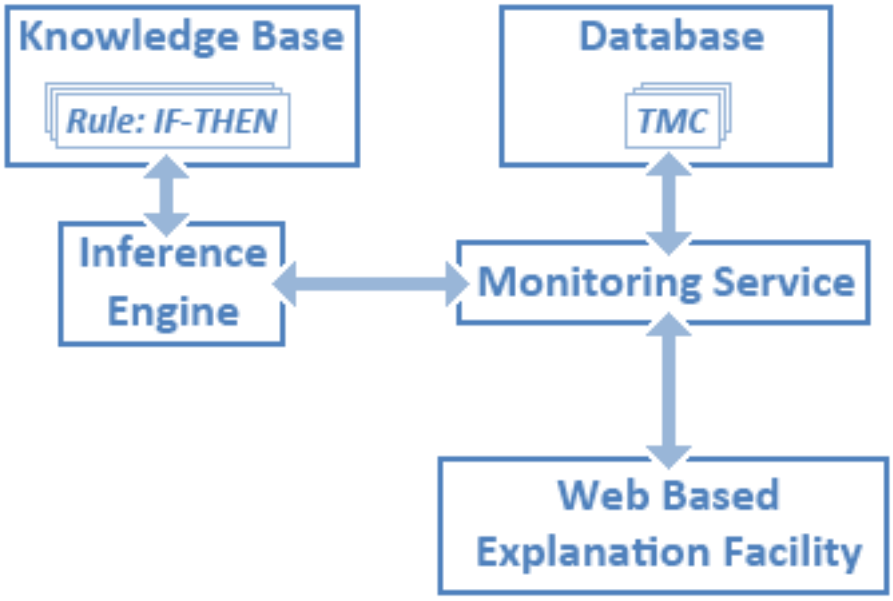


Figure 5.1.2.1 – KB System Structure

1. **Knowledge-Base** – For this project, the knowledge-base is the same as in a basic knowledge-based expert system and it contains the experts knowledge represented in a form of rules as well as some meta-data regarding those rules and their execution. The design of the knowledge-base is explained further in section 6.
2. **Database –** the database normally includes sets of facts and other information to be matched in the IF part of the rules, however, in this project the database only contains information about the TMC machines and some configuration data. The facts and information used in the IF part of the rules is taken from the TMC machines directly.
3. **Inference Engine –** The inference engine in this project preforms the same role as in a basic knowledge-based expert system and carries out the reasoning process using the information from the TMCs and rules from the knowledge-base.
4. **Monitoring Service –** this is an additional component, usually not present in a basic knowledge-based expert system and it is the "driver" of the operation. It is in charge of collecting data from the Database and gathering the state of the TMCs. It then uses the Inference Engine to detect faults, saves the results and issue notifications to the end-users. The design of this monitoring service is explained in detail in section 6.

5. **Explanation Facilities** – allows the user to see how and why a particular conclusion has been reached. In this project the Explanation Facility is a web based solution and it is also acts as the **User Interface** as it is the only interface for end-users to use in this system.

In a usual knowledge-based expert system, the system interacts with end-users which give it information and expect results; however this system is a combination of a few sub-systems interacting with each other. The end-users have very limited use of the system and only need it to view the state of each TMC and the results of past executions by the monitoring service, after being notified of a fault.

5.2 FDI Approach

A fault is defined as “an unpermitted deviation of at least one characteristic property or parameter of the system from the acceptable/usual/standard condition” (Sampath et al 1996, Isermann, 1997). Based on that definition, we introduce the term “***fault case***” which means a possible case or scenario in which a fault might be present, and it is essentially a group of attributes or parameters and their acceptable/fault-free condition.

The FDI approach used in this project is a hybrid of the Model-Based and Signal-Processing Based approaches. The FDI engine analyses the state of a TMC represented by a JSON string. In a “pure” Model-Based approach (Figure 5.2.1) the current TMC state JSON string would be converted to a current “state model” and compared to a complete “fault-free state model”. Residuals (differences) will be generated and then analysed to detect faults.

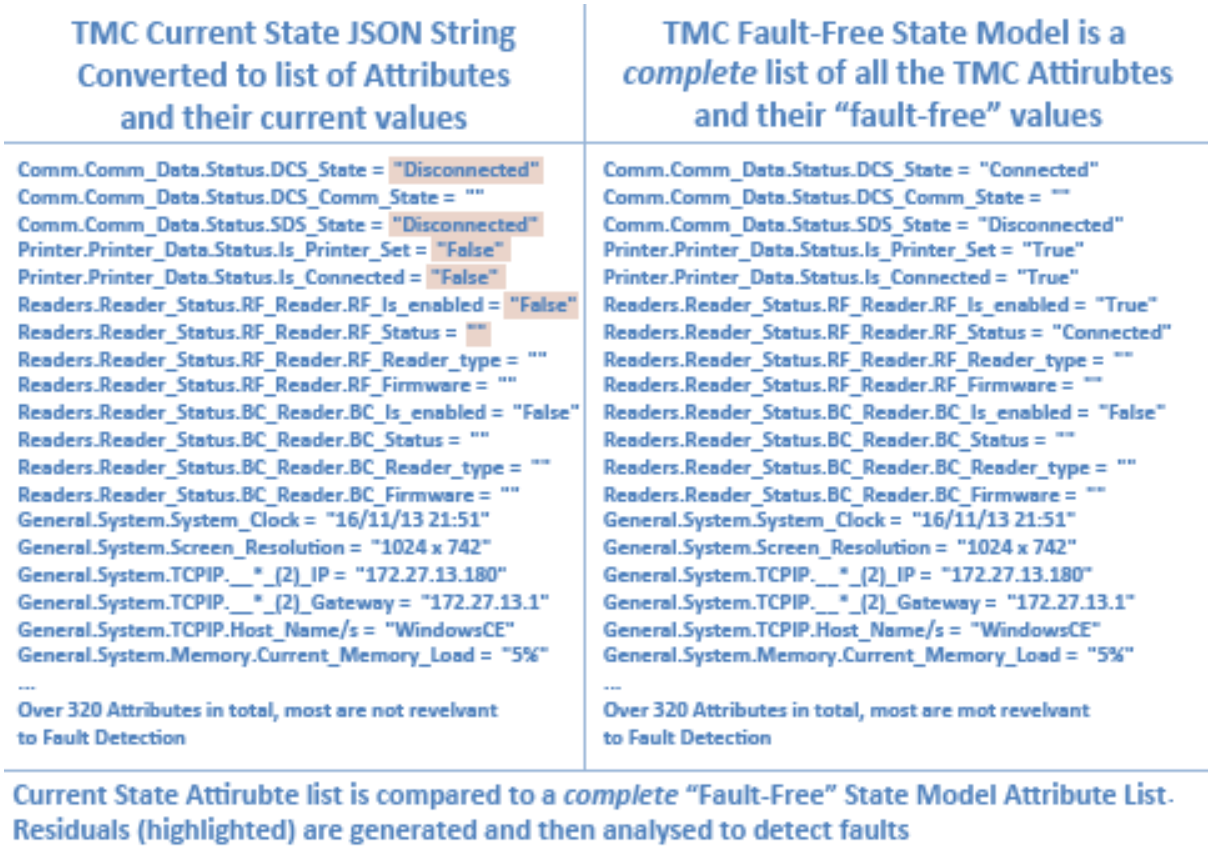


Figure 5.2.1 – Example of a TMC Current State Attribute List and a complete “Fault-Free” Attribute List Model.

Since there is no complete model of a fault-free TMC state, a system which by default can be configured differently for each specific client, sub-models of different attributes of a TMC state

model were defined (Figure 5.2.2). These sub-models were grouped together by possible faults they might indicate, termed "***fault cases***", as defined above.

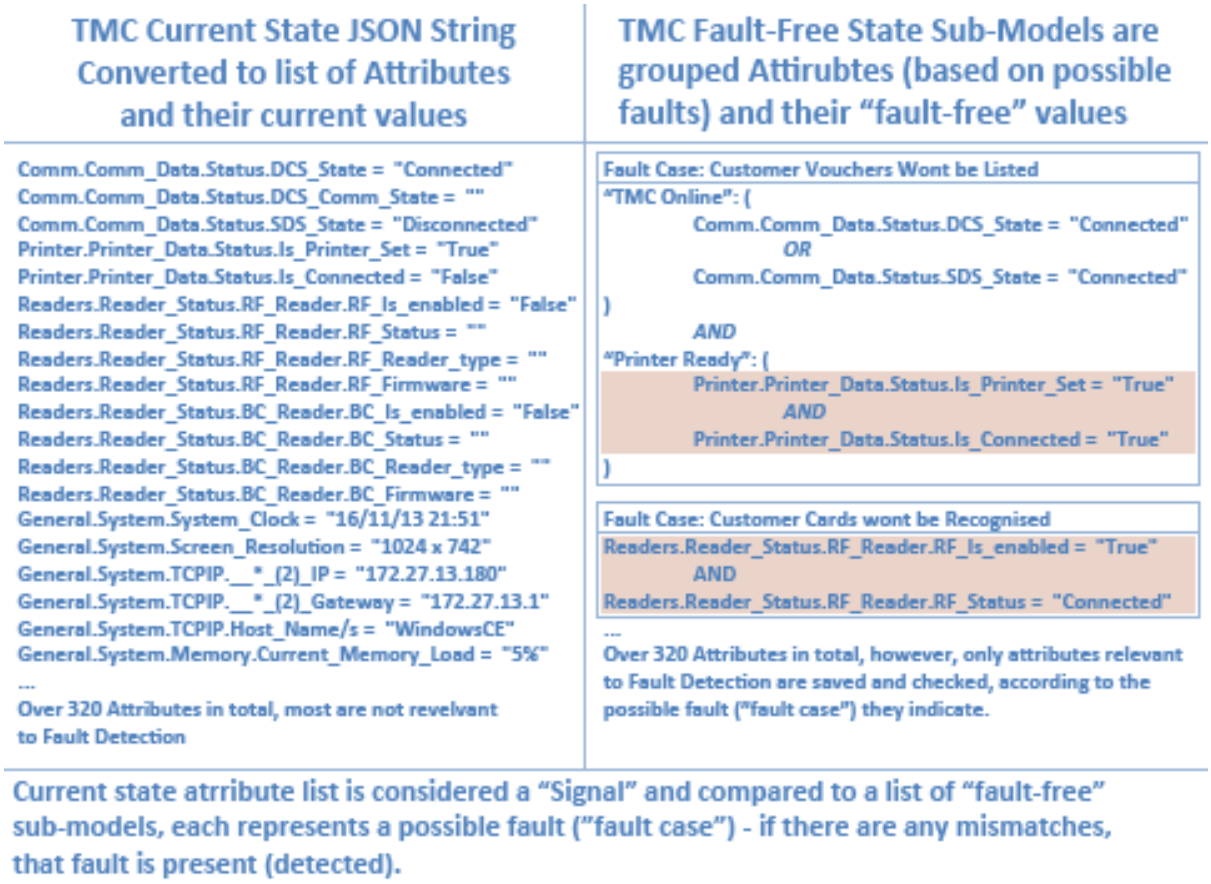


Figure 5.2.2 – Example of a TMC Current State Attribute List and a "Fault-Free" Attributes Sub-Models based on the Fault they indicate.

In this hybrid approach, the residual generation is not done by comparing the state of the TMC to a fault-free state model, but the current state is the residual (and there-fore considered a "signal"). The signal is analysed using a collection of "***fault cases***", each based on a sub-model of a fault-free TMC state attributes and has a set of tests to determine if a fault is present (Table 5.2.3).

Fault Case	Tests to see if present

Customer Vouchers Won't be Listed
 	
("DCS" and "SDS" are different Fortress Servers that TMCs can connect to)
	• Is Printer Ready?
	  ○ Is Printer Set? And,
	  ○ Is Printer Connected?
	• Is TMC "online"?
	  ○ Is DCS Connected? Or,
	  ○ Is SDS Connected?

Customer cards won't be recognised	• Is RF Reader Configured? • Is RF Reader Connected?

Table 5.2.3 – Example of possible TMC Fault Cases

The fault-case tests employ rules from the knowledge-base. The rules are based on TMC state attributes and their *fault-free acceptable* value. The tests compare the values of certain attributes from the current state model, to their acceptable value in a fault-free TMC state model. If there are mismatches, the fault-case has been proven and a fault detected (Table 5.2.4).

Fault Case Test	Test Rule(s)
Is Printer Ready?	**IF** *"Printer.Printer_Data.Status.Is_Printer_Set"* is *"True"* **AND** **IF** *"Printer.Printer_Data.Status.Is_Connected"* is *"True"* **THEN** "Not Present".
Is TMC "online"?	**IF** "*Comm.Comm_Data.Status.DCS_State*" is "*Connected*" **OR** "*Comm.Comm_Data.Status.SDS_State*" is "*Connected*" **THEN** "Not Present"

Table 5.2.4 – Examples of Fault Case Tests and their rules representations.

This answers the issue of the dynamic nature of the system, and the lack of a complete fault-free TMC state model.

5.3 Software Engineering Method

In order to achieve this solution, the system was split into a few sub-systems described in Figure 5.3.1.

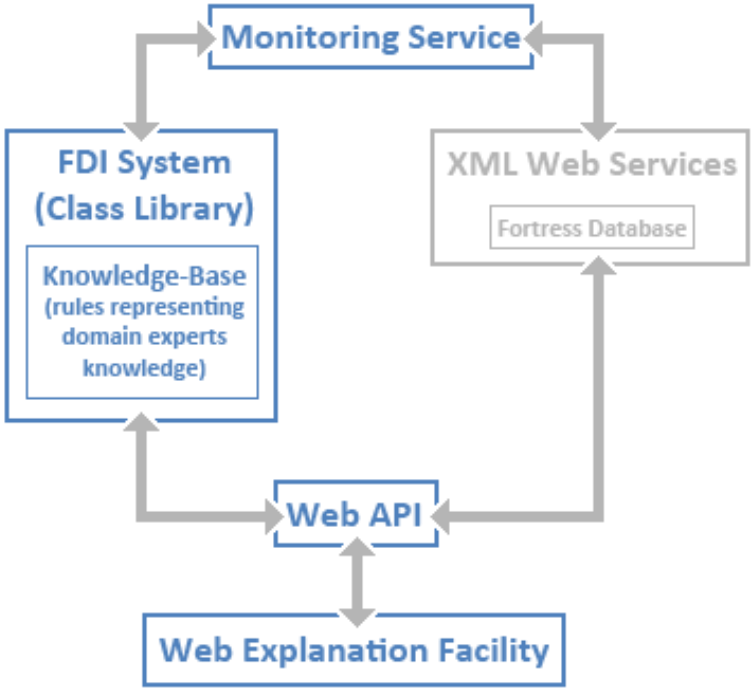


Figure 5.3.1 – The sub-systems and their interaction.

- ***Fault Detection (FDI) System*** – an inference engine which detects faults by using a knowledge-based system that contains knowledge about possible faults in a TMC. This knowledge is represented in a form of rules acquired from the TMC developers (i.e. "domain experts").
- A **monitoring service** - using Fortress's XML Web Services, gets a list of all TMCs (and their current state) and runs fault detection tests using the FDI system and, sends notifications when needed.
- **Web API** that provides the current state of the system and each TMC the service monitors.
- A **web-based explanation facility** which will be used by both hardware/software support teams to check the state of the system and resolve faults.

There are different ways to implement these sub-systems, and the approach taken in this project was dictated by the organisational needs and the existing systems. Since the development environment in Fortress is based on Microsoft .NET and MS-SQL server, we decided all the above sub-system will be Windows based and written in C# using MS-SQL and the .NET framework.

5.3.1 Software Development Methodology

The software development methodology used in Fortress is a very high-level Agile approach, a mixture of Scrum and RAD (Rapid Application Development).

The time constraints of the project, the need for a solution that could be used internally by Fortress and the fact the author of this project needed to work with other developers in the company (domain experts), led us to adopt the Dynamic Systems Development Method.

5.3.1.1 Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) is a management and control framework for rapid application development (RAD). The RAD approach is designed to enable delivery of a working solution rapidly. DSDM is an industry standard definition of the RAD process, which defines a structure and controls to be use in Agile/RAD projects, without specifying a methodology (Bennet et al, 2006).

DSDM is a proven framework for agile project management. It's designed to help deliver results quickly and effectively. Its main concern is the strategic goals and incremental delivery of actual business benefits while enabling control of time, cost, risk and quality (Craddock et al, 2012)

5.3.1.2 DSDM Atern

The DSDM Atern is a framework based on best practices, aiming to provide a flexible and controlled process to deliver solutions while effectively using people's knowledge, development techniques and modelling to achieve efficient delivery times (DSDM Consortium, 2014). Its underlying philosophy is that projects must be aligned and clearly fit a defined strategic goal, focusing on quick delivery of real business requirements. To support this philosophy, there are 8 guiding principles detailed in Figure 5.3.1.2.1 below (DSDM Consortium, 2014):

Figure 5.3.1.2.1– The 8 principles of DSDM Atern (based on DSDM Consortium, 2014)

5.3.1.3 DSDM Agile Project Life-Cycle

According to the fifth and sixth principles described above, the Atern life-cycle is iterative and incremental. The iterative concept enables to examine the work being done, comment on it and change or refine requirements during development of the next increment.

The project life-cycle (Figure 5.3.1.3.1) has four main phases: Feasibility, Foundations, Evolutionary Development and Deployment. These are usually preceded by a Pre-Project phase and Post-Project phase. The Pre-Project phase goal is to make sure that only the right projects get started and set up correctly, the Post-Project phase is designed to support and keep the delivered solution operating and validating it answered its designed business needs. The four core phases are described below (Craddock et al, 2012):

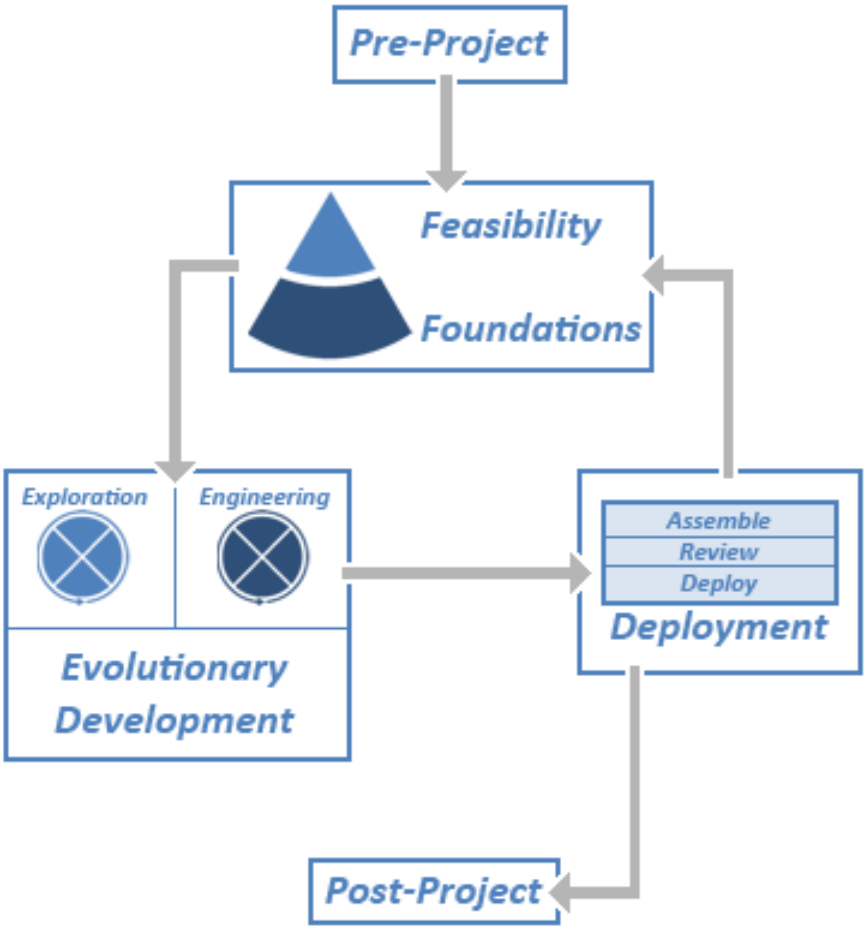


Figure 5.3.1.3.1 – DSDM Project Life Cycle (based on Craddock et al, 2012 and DSDM Consortium, 2014)

1. **Feasibility** – designed to evaluate by high-level investigation of the proposed solution, costs and timeframes if a project is viable.
2. **Foundations** – aimed at building firm and long-lasting foundations and infrastructure for the project.
3. **Evolutionary Development** – used to iteratively and incrementally investigate the requirements and transfer them to a viable solution. This phase can be divided into 2 sub-phases (DSDM Consortium, 2014):
    a. ***Exploration*** – Building none production-ready deliverable focused on validating it can deliver what is needed, while answering changing requirements and overall needs.
    b. ***Engineering*** – evolve the deliverable from the Exploration phase to achieve a production ready solution.
4. **Deployment** – designed to bring the production-ready solution, produced in the 3rd phase to live use. Additionally, in this phase the solution is reviewed, prior to actual deployment.

As detailed below, in section 5.3.2, the five stages of the work plan proposed for this project were roughly aligned with the four core phases of the DSDM Agile Project Life-Cycle (with the project proposal considered as Pre-Project and the 5th stage, Report, considered as Post-Project).

The details of how each of the core 4 phases was used in the Development Process of this project is explained in section 6.

5.3.1.4 MoSCoW Prioritisation

In order to be able to deliver on established fixed dates, and adhering to a fixed cost, some work might have to be deferred, alternatively, to achieve business needs, some not originally planned work might be required. Essential deliverables must be completed and only less crucial deliverables maybe be disregarded. MoSCoW prioritisation is an efficient method to achieve these guidelines, and it dictates the following:

- **Must Have** – fundamental requirements for the solution to be acceptable and answer business needs. These define a "Minimum Usable Subset" which the project guarantees to deliver.
- **Should Have** – important requirement that should be delivered if time permits, however business needs could still be met without them
- **Could Have** – requirements which can be easily disregarded and are less valuable.
- **Won't Have** (this time) – requirements that may be included in future development and are excluded from plans for current delivery.

In DSDM, MoSCoW is used to provide the base for decision making regarding team activity at all levels. It enables business expectations and needs to be defined at a project level and ensure the project will deliver the *Must Haves*, will likely deliver some of the *Should Haves* and may deliver some of the *Could Haves* (Craddock et al, 2012).

5.3.2 Work Plan

The implementation and development of the system was split into the following stages:

5.3.2.1 Stage 1 – Preparations and Knowledge Acquisition (November, 2013):

The first stage of the development was to collect information about the system, the expectations of the end-users and knowledge regarding the rules it will employ to detect faults. This involved:

- Interviewing and surveying the TMC developers (domain expert)
- Review the TMC code to gather the information needed to design the rule-based knowledge-base.

- Interview the support teams and collect feedback on different issues and faults usually detected in “real life” environment. Since they are the end-users, they were also asked to provide suggestions on how they would like the system/interface to work.

5.3.2.2 Stage 2 – Specifications, Documentations and First Implementation (December, 2013):

The collection of the information in Stage 1 enabled establishing a conceptual idea of the system. With that, the following steps were taken to begin the design and development of the system.

- Write up specification documents and testing documents.
- Building the first version of the knowledge-base, inference engine and monitoring service.
- Building a “tester app” to test the rules, the inference engine and simulate interaction with TMCs. More details about this approach are given in section 6.

5.3.2.3 Stage 3 – Analysis, Further Development and Alpha Testing (January, 2014):

Once the initial version was built, based on information gathered in Stage 1 and documentation from Stage 2, further development steps were taken:

- Testing the monitoring service and inference engine, refining the rules in the knowledge-base and establishing a larger set of rules.
- Implementing the notification mechanism.
- Building the Web API that provides the current state of each TMC the service monitors.
- Building the web-based explanation facility which will be used by the end-users.

5.3.2.4 Stage 4 – Verification, Validation and Beta Testing (March, 2014):

- Additional testing the system, revising and adapting the solution, the inference engine and rules accordingly.

5.3.2.5 Stage 5 – Report (April, 2014):

- Prepare the final report for submission.

6. Design and Implementation

6.1 Overview

In this section the design and implementation process of this project is described. Details about how DSDM was used and the different development technologies employed in this solution are also provided.

Additionally, this section describes the systems architecture and the design choices that were made prior and during development, and the development process itself.

6.2 Development Environment and Technologies

This section details the software tools, techniques and technologies used in this project.

6.2.1 Microsoft .NET Framework

The .NET Framework is a managed execution environment that provides a wide range of services to its running applications. It consists of two major components: *the common language runtime* (CLR) and the *.NET Framework Class Library* (MSDN, 2014).

- ***The Common Language Runtime (CLR)*** – a run-time environment which runs the code and provides services such as memory management (and garbage collection), structured exception handling, and explicit free threading enabling multithread scalable applications (MSDN, 2014).
- ***The .NET Framework Class Library*** – a collection of classes which include interfaces and value types that aim to accelerate and optimize the development process. These represent the foundation on which .NET applications, components and controls are built on. The framework provides an extensive set of interfaces, abstract and concrete classes to enable faster, structured and expandable development process (MSDN, 2014) .

This solution is written in C# (CSharp), a type-safe object-oriented language that runs on the .NET framework (MSDN, 2014) and employs many of the .NET framework classes library components, such as ***System.Collections*** (classes that define collections objects, such as List and Dictionary) and ***System.Reflection*** (used to examine and manipulate instances of loaded types).

6.2.2 Microsoft .NET XML Web Services

XML Web services are programmable services that provide different elements of functionality and are accessible to a number of systems using standards such as XML and HTTP. They provide a viable solution for enabling data and system interoperability and can be accessed by a single application or a number of applications using TCP/IP (MSDN, 2013).

The XML Web Services used in this solution are general Fortress Web Services ***(not written specifically for this project),*** and are used to collect and save information from and to the databases.

6.2.3 ASP.NET Web API

ASP.NET Web API is a framework designed to allow quick and easy development of HTTP services that reach a wide range of clients (such as browsers and mobile devices). ASP.NET Web API is a platform for building RESTful (Representational State Transfer) applications on the .NET Framework. (Microsoft, 2014)

In this project, Web API is used by the web based explanation facility, to get information about past executions by the FDI system.

6.2.4 Microsoft SQL Server

Microsoft SQL Server (MS-SQL) is a rational database management system (RDBMS). Its primary function is to store and retrieve data as needed by different applications. It uses Transact-SQL (T-SQL), an extension the Structured Query Language (SQL), and it enables running C# code (using CLR) directly from the SQL Server (MSDN, 2014).

All the information and data (such as rules, execution results and other meta-data) used in this solution are stored on different databases using Microsoft SQL (MS-SQL) Server.

6.2.5 JavaScript Object Notation (Newtonsoft Json.NET)

JavaScript Object Notation (JSON) is a light-weight text format which uses structured data to allow interchange between different programing languages (EMCA International, 2013). JSON is language independent, “self-describing” and easily readable. It is smaller than XML, faster and easier to parse (W3Schools.com, 2014).

Newtonsoft Json.NET is an open-source, high-performance JSON parser for the .NET Framework, used for serializing and de-serializing .NET objects to and JSON text (James Newton-King, 2014).

In this solution, JSON is used for representing the status of a TMC machine, and in the Web API and web-based explanation facility.

6.3 Architecture and Design Choices

6.3.1 Architecture

The overview of the system was shown in Figure 5.3.1 above. Figure 6.3.1.1 details the systems components, and their interaction in a single Fault Detection execution.

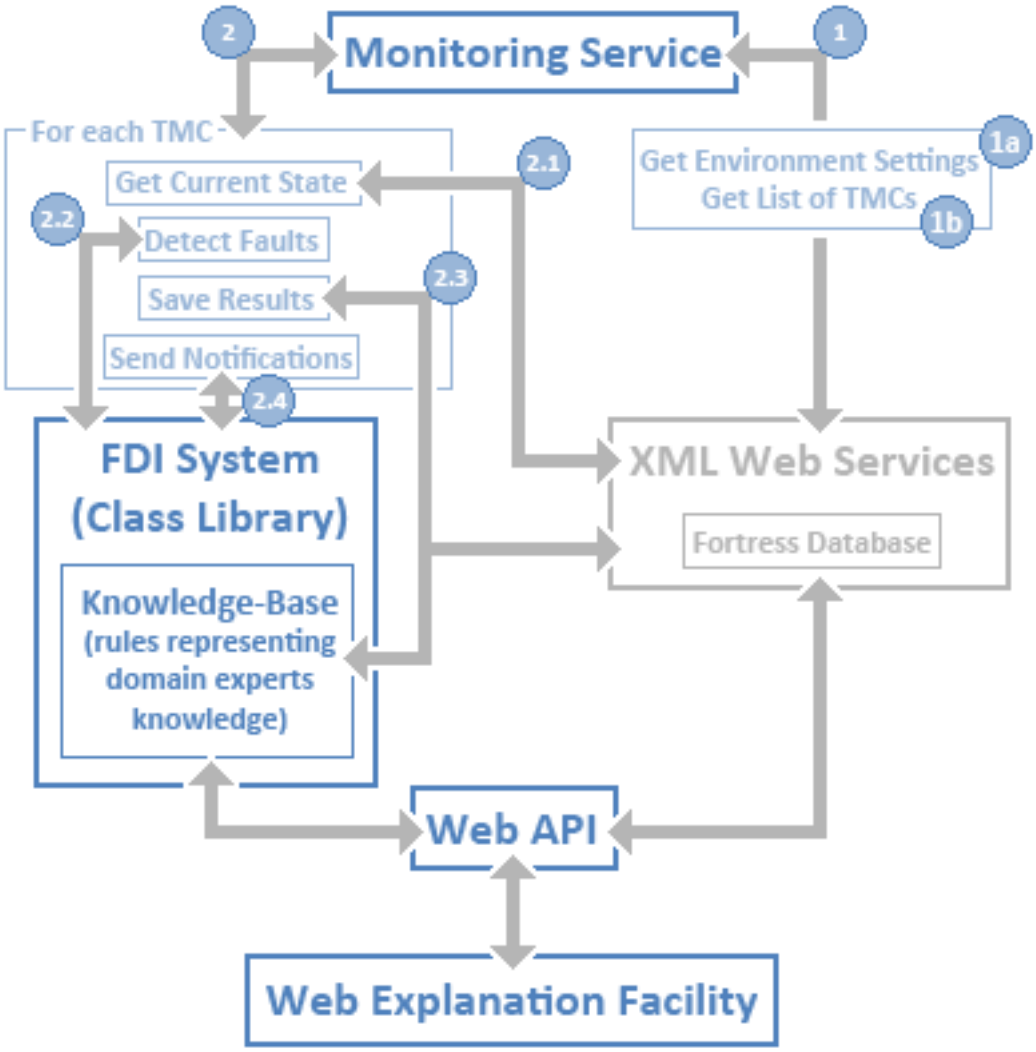


Figure 6.3.1.1 – A Detailed Fault Detection Cycle (i.e. a Single Execution by the Monitoring Service)

- **The rule-based knowledge-base** – contains sets of possible faults ("***fault-cases***") and their corresponding tests ("***fault-cases tests***") and rules based on the knowledge of the TMC developers (domain experts), as defined and described in section 5.
- **The FDI System** – a Class Library (DLL) that infers the state of each TMC using a set of fault-cases and their corresponding tests and rules, according to the FDI Approach described in section 5.

- **The Monitoring Service** – periodically (on a configurable timed schedule) preforms the following:
    - Using the **Fortress XML Web Services** (1):
        - Loads the information about the execution environment (1a).
        - Gets a list of all TMCs in the venue/environment (1b).
    - For each TMC (2):
        - Gets current State (2.1)
        - Uses the fault detection inference engine to get a collection of fault-cases and analyse the states to detect faults based on the fault-cases tests, environment information and other meta-data (2.2).
        - Saves the results of each execution (2.3) to the knowledge-base to be used by the Web API and explanation facility and attaches it to the TMC in the Fortress Database (using the XML Web Services).
        - Sends notifications (2.4) based on the results of the fault detection execution.
- **Web API** – Uses the XML Web Services and the FDI system to provide the current state of each TMC the service monitors based on past fault detection executions.
- **A web-based explanation facility** - uses the Web API to provide an overview of the current state of each TMC and the results of the past fault detection executions.

6.3.2 Constraints

Since the proposed solution was to be a part of the Fortress system, some constraints were enforced and taken into consideration when designing the system:

- **The Framework, the Development Environment and Database Server (C1)** – In order to provide a solution which can be maintain by other developers in the company, the framework (Microsoft. NET), development environment (Visual Studio) and Database Server (Microsoft SQL Server) were used in the development of the solution.
- **XML Web Services (C2)** – the use of XML web services was also enforced to keep this solution in-line with all other Fortress projects.
    - **Parameter-less Constructors (C2a)–** a derived constraint of using XML Web Service was that in order to be serialize-able, all classes consumed by such web service must have a parameter-less constructor (MSDN, 2014).

- **Modularity and Expendability (C3)** – one of the key development concepts in Fortress is that requirements constantly change. The designed system had to be flexible, modular and expandable to allow quick reply to changing requirements.
- **None-Dependency and Self-Maintainability (C4)** – the key constraints of the project is that this project development should not interfere with the development of other projects (i.e. the TMC project).
    - **None-Dependency (C4a)** - when building new versions of the TMC, the developer should not be aware of the FDI system. The TMC development should stand on its own without any consideration or changes to allow faults to be detected.
    - **Self-Maintainability (C4b)** – the FDI system should be maintained on its own and should be able to support current (and new versions) of the TMC without any maintainability or code changes.
- **Accessibility (C5)** – the support teams needed to be able to view the state of each TMC in clients' environment (venues) from the main Fortress offices and while on-site.

Some of the above constraints (C1, C2) were resolved before implementation began. The following sections detail how all other constraints (C3 to C5) were addressed and resolved.

6.3.3 Design Choices

After several discussions with the TMC developers, the general structure and design of the system (Figure 6.3.1.1) was established. However, during those discussions certain constraints were discovered (detailed above) and needed to be addressed before development began.

This section highlights some design choices and how they address and resolve these constraints.

6.3.3.1 JSON Strings

One of the key constraints of this project, and the most important one was that this project will be self-sustained (C4b), will not affect the development of the TMC or be dependent on it (C4a), or need to be updated as new versions of TMC come out. In order to address this constraint, we decided that the TMC would return its state as JSON String, taken directly from its UI controls. This means that once the developers of the TMC add functionality, or remove certain element from the system, it will automatically be represented in the state string sent back to the service.

This solution however presented another constraint in which all element of the TMC status are represented as String types, even if they are Integers, Booleans or Date objects. This derived constraint was solved by using Generics and Regular Expressions.

6.3.3.2 Generics and Regular Expressions

Generics enable developers to define type-safe data structures, without committing to actual data types and in effect they enable reuse of data processing algorithms without duplicating type-specific code (MSDN, 2014). C# Generics are similar to C++ Templates as both enable using parameterized types, however C# Generics do not try to offer all the functionality C++ templates provide. Additionally, in C# generic types are substituted at runtime and are preserved to instantiated objects (MSDN, 2014).

Regular expressions provide a powerful and efficient method for processing text. Using extensive pattern-matching notation, regular expressions enables developers to quickly parse text to find specific patterns and validate the text to ensure that it matches a certain pattern, for example if a text represents a Date (i.e. “01/01/2014”, or “01-01-2014”), or a certain time in a day (i.e. “10:00:00” or “10.00am”) (MSDN, 2014).

Using both these powerful tools, the string representation of objects from the TMC were parsed and turned into type-safe objects, which were then compared and analysed easily. When the JSON String representing the TMC state is parsed, different attributes are converted into “computed types” using Regular Expression:

```
private static FaultDetectionParamType getTypeFromValue(string value)
{
	Match m = Regex.Match(value, @"^(\d*?)$");
	if (m.Success)
	{
		this.computedType = FaultDetectionParamType.INTEGER;
	}
}
```

Figure 6.3.3.2.1 – Pseudo-code method to get FaultDetectionParamType from a String

The above pseudo-code (Figure 6.3.3.2.1) uses regular expressions to check if a given string (value) only contains digits. If that is the case, it presumed to be an integer. Using regular expressions, string representations can be analysed and converted to type-safe types.

This parameter can then be retrieved externally using generics (Figure 6.3.3.2.2), without the consumer knowing what type it really is:

```
public T get<T>()
	if (this.computedValue.Length > 0
		&& this.computedType != FaultDetectionParamType.NULL
		&& this.computedType!= FaultDetectionParamType.UNKNOWN)
	{
		return (T)FaultDetectionParam.parse(this.computedValue,
		this.computedType);
	}
	return default(T);
}
```

Figure 6.3.3.2.2 – Pseudo-code method using Generics to convert string to its type-safe real type.

Once these attributes are "converted back" to their "real" type-safe types, they are used by the inference engine and compared to expected values from the rules in the knowledge-base according to the FDI approach described in section 5 (Figure 6.3.3.2.3).

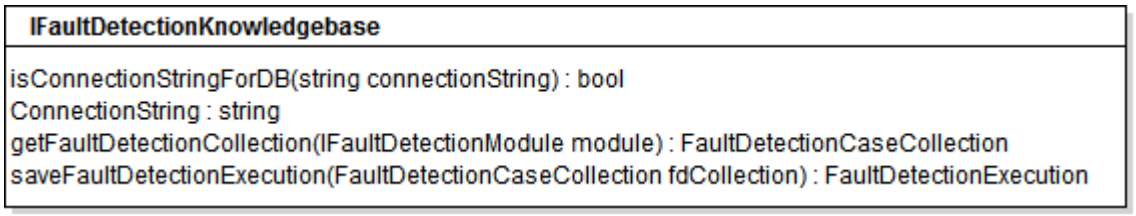
```
public FaultDetectionResult evaluate(FaultDetectionParam expected, FaultDetectionParam
tmc)
{
	switch (expected.ComputedType)
	{
		case FaultDetectionParamType.STRING:
			if (tmc.ComputedType == FaultDetectionParamType.STRING)
			{
				return new
				FaultDetectionResult(expected.get<String>().Equals(tmc.get<Str
				ing>()));
			}
			return FaultDetectionResult.invalidParameterOperator(expected, tmc,
			this);
	}

	return new FaultDetectionResult(false, "Paramter 1 (" + expected.Type + " == " +
	expected.ComputedType + ") invalid for equality operation");
}
```

Figure 6.3.3.2.3 – Pseudo-code method using Generics to compare the value from the TMC current state and the expected value from the knowledge-base.

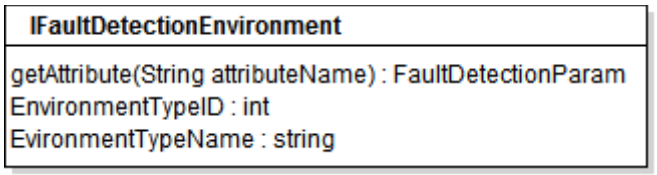
6.3.3.2 Interfaces

All non-concrete or flexible parts of the system were designed based on interfaces. Standardisation of messages and interfaces is a model way to provide stable, technology agnostic mechanism for communication and interoperability of systems in complex architectures (Corsello, 2008). Interfaces enable greater modularity and expendability and address constraint C3. A few key components of the system were designed as interfaces and have one or more concrete classes implementing them.

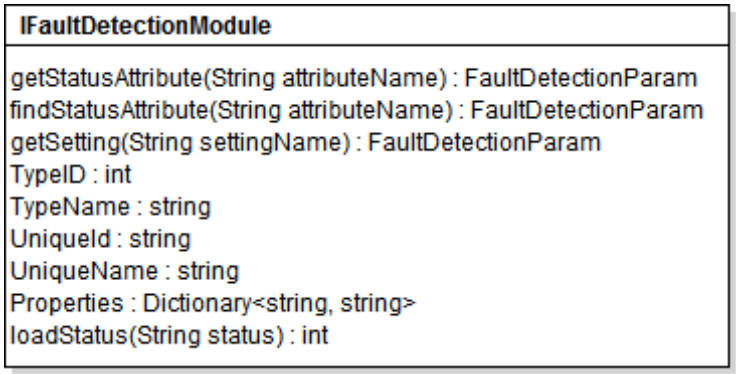
- **IFaultDetectionKnowlegebase** – the Fault Detection (inference) Engine can use different knowledge-base types, from MS-SQL, to SQL Azure and NoSQL databases. The engine does not "care" where the knowledge is stored, and it uses an interface to get the collection of fault-cases, and their corresponding tests and rules, notifications information, etc.



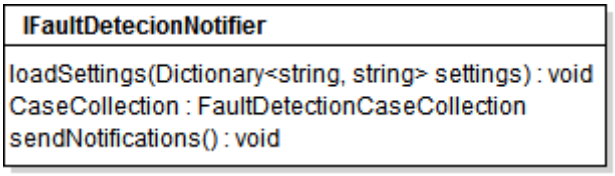
- **IFaultDetectionEnvrionment** – similarly, the engine can work in multiple Fortress environments (i.e. Sport Venues, Universities, etc.). It does not need to know which environment it's running at, as long as it's able to get certain attributes about the environment in order to detect faults.



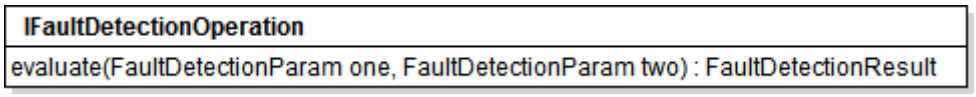
- **IFaultDetectionModule** – although the proposed solution is for a Fault Detection and Isolation for TMCs, to enable more expandability, we decided that it should be possible use the engine to detect faults in other Fortress modules. The module interface allows expanding the usability of this solution greatly; *to virtually any fortress component that uses its propriety TCP/IP protocol.*



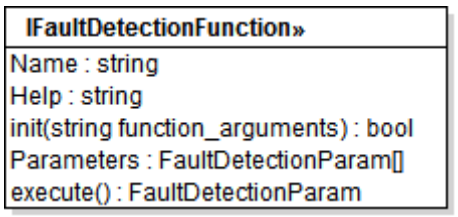
- **IFaultDetectionNotifier** –the idea behind the IFaultDetectionNotifier interface is to allow Fortress to change the way support engineers get notification, for example, in the future notifications might be sent via SMS instead of Emails.



- **IFaultDetectionOperation** – this basic interface was designed to allow expanding the rule-base and more specifically, rule types. During the development of this project, it became noticeable that there are different types of operations, i.e. "Equality" ("X" equals "X"), or "Contains" ("XYZ" contains "Y") etc. Using an interface for different rule operation means that Fortress can expand the rule base and create new type of rules as it see fits.



- **IFaultDetectionFunction** – As described above, the state of the TMC is given in a JSON string, and some of the attributes are mixed values that only certain part of them is needed. This introduced the need for functions to extract certain information from the status of the TMC. To follow suit with the rest of the project, the function interface was designed to enable creating new functions quickly and easily.



All the above interfaces can be seen in the code listings in Appendix B.

6.3.3.4 Reflection and Namespaces

Reflection in .Net/C# enables developers to get information about loaded assemblies and the types defined in them (classes, interfaces, etc.). Reflection is used create type instances at run time (MSDN, 2014).

Namespaces in .Net/C# are used to create a scope that contains related objects and are used to organize code elements and create globally unique types (MSDN, 2014).

In this project, all concrete classes that implement ***IFaultDetectionOperation*** and ***IFaultDetectionFunction*** interfaces (described above) were grouped together in ***FaultDetection.Operations*** and ***FaultDetection.Functions*** namespaces respectively. When an

instance of the Fault Detection Engine is created, all the classes in these namespaces are loaded into specific lists dynamically at runtime using Reflection (Figure 6.3.3.4.1).

```
private void loadOperations()
{
	Assembly pAssembly = Assembly.GetAssembly(typeof(FaultDetectionEngine));
	if (pAssembly != null)
	{
		// searching for all the types in "FaultDetection.Operations" name space
		foreach (Type pType in pAssembly.GetTypes().Where(t
		=>String.Equals(t.Namespace, "FaultDetection.Operations",
		StringComparison.Ordinal)).ToArray())
		{
			Type pInterface =
			pType.GetInterface(typeof(IFaultDetectionOperation).ToString());
			// if this type implements the IFaultDetectionFunction interface, it
			will not be null
			if (pInterface != null)
			{
				// creating a new IFaultDetectionOperation object and adding
				it to the Loaded Operations List.
				ConstructorInfo pConstructor = pType.GetConstructor(new Type[]
				{ });
				IFaultDetectionOperation pInstance = pConstructor.Invoke(null)
				as IFaultDetectionOperation;
				if (pInstance != null)
				{
					_loadedOperations.Add(pInstance);
				}
			}
		}
	}
}
```

Figure 6.3.3.4.1 – Pseudo-code of loading IFaultDetectionOperation array using Reflection.

These lists are then used in the inference process to compare values or execute functions when needed (Figures 6.3.3.4.2 and 6.3.3.4.3).

```
private IFaultDetectionOperation getOperation(string operationTypeName)
{
	// checking if any operation have been loaded...
	if (_loadedOperations.Count > 0)
	{
		foreach (IFaultDetectionOperation operation in _loadedOperations)
		{
			if
			(operation.GetType().Name.ToLower().Equals("faultdetectionoperation"
			+operationTypeName.Replace("_","").ToLower()))
			{
			    return operation;
			}
		}
	}
	return null;
}
```

Figure 6.3.3.4.2 – Pseudo-code of getting an IFaultDetectionOperation from a list of loaded operations.

```
private FaultDetectionResult execute(FaultDetectionParam one, FaultDetectionParam two,
string operationTypeName)
{
	IFaultDetectionOperation operation = getOperation(operationTypeName);
	if (operation != null)
	{
		return operation.evaluate(one, two);
	}
	else
	{
		return new FaultDetectionResult(false, "Invalid Operation: " +
		operationTypeName + " (" + operation + ")");
	}
	return new FaultDetectionResult(false);
}
```

Figure 6.3.3.4.3 – Pseudo-code of using IFaultDetectionOperation interface to evaluate two FaultDetectionParams.

This mechanism gives extreme modularity and flexibility. If a new operation type is needed, all that is required from the developer is to create a new concrete class (implementing the ***IFaultDetectionOperation*** interface) under the ***FaultDetection.Operations*** namespace, and it can be used instantly.

6.3.3.4 Web API and Web-Based Explanation Facility

As detailed above, constraint C5 imposed the need for the end users (i.e. Fortress support teams) to be able to view and monitor the state of each TMC in clients' environment (venues) from the main

Fortress offices and while on-site. In order to achieve that, we decided that execution results and the state of the system will be available via a Web API and a Web Based explanation facility which consumes that API and is accessible to end-users.

6.4 Development Process

6.4.1 Phase 1 - Feasibility

The first stage of the development was to gather as much information about the solution and come up with an execution plan. It involved:

- Interviewing and surveying the TMC developers (domain expert)
- Review the TMC code to gather the information needed to design the rule-based knowledge-base.
- Interview the support teams and collect feedback on different issues and faults usually detected in "real life" environment.
- Collect requirements/expectations from end-users about how they would like the system/interface to work.

After the 1st iteration of interviews, a basic overview of the system components and their responsibility in the solution was established (Table 6.4.1.1):

Component/Actor	Task List/Responsibility
Monitoring Service	• Get List of TMCs • Get Environment Information • Load Notification Settings (i.e. SMTP settings) • Get State of each TMC from the list. • Send Notifications (when needed) • Save Fault Detection Execution Results for each TMC.
Fault Detection (inference) Engine	• Get a Collection of Fault Cases (including their tests, rules and notification details) • Execute Fault Detection.
Web-Based Explanation Facility	• Display List of TMCs and their current state. • Enable viewing past Fault Detection Executions.
End-User	• View state of each TMC (via the Explanation Facility)

Table 6.4.1.1 – Component Task List

As explained in section 4, in a KB Expert System, the knowledge-base holds the domain expert knowledge, represented in rules. After interviewing the TMC developers, it was decided that the Fault Detection (inference) Engine would use ***backwards chaining*** and would go through a list of

possible faults (hypothesises), termed "***Fault Cases***". Each Fault Case can be established as "Not Present" (i.e. "Passed") once a set of rules, termed "***tests rules***", were checked.

According to the FDI Approach described in section 5, each test goal is to examine one or more ***Parameters*** or ***Attributes*** (facts) from the TMC State String, and compare its value to what it would be in a fault-free TMC. Tables 6.4.1.2a and 6.4.1.2b describe a how Fault Cases were defined:

Fault Case Description **(Suggested by the Domain Expert)**	TMC Offline
Fault Case Tests **(Defined by the Domain Expert)**	**1. Check if TMC Online** • Is the TMC connected to a DCS? • Is the TMC connected to an SDS? *"DCS"* and "*SDS*" are two different prosperity Fortress Servers the TMC can communicate with.
Resulting Rule **(Translated by the Knowledge Engineer)**	***IF*** *(Comm.Comm_Data.Status.DCS_State is "true")* ***OR*** *(Comm.Comm_Data.Status.SDS_State is "true")* ***THEN*** *"Fault Not Present".*

Table 6.4.1.2a – Example of a simple Fault Case

Fault Case Description **(Suggested by the Domain Expert)**	Customer cards won't be recognised
Fault Case Tests **(Defined by the Domain Expert)**	1. Check RFID Reader • Is RFID Reader Configured? • Is RFID Reader Connected? 2. Check Barcode Reader • Is Barcode Reader Configured? • Is Barcode Reader Connected? 3. Check Encryption Keys • Are Encryption Keys Correct?
Resulting Rule **(Translated by the Knowledge Engineer)**	**IF (** **(**Readers.Reader_Status.RF_Reader.RF_Is_enabled is "*True*" **AND** Readers.Reader_Status.RF_Reader.RF_Status is "*Connected*"**)** **OR** **(**Readers.Reader_Status.BC_Reader.BC_Is_enabled is "*True*" **AND** Readers.Reader_Status.BC_Reader.BC_Status is "*Connected*"**)** **)** **AND** (Readers.CKKeyString *EQUALS* Environment.CKKeyString **AND** Readers.3DESKeyString *EQUALS* Enviornment.3DESKeySting**)** **THEN** "*Fault not Present*"

Table 6.4.1.2b – Example of a complex Fault Case

Through an iterative process a large set of fault-cases was created, and then through generalisation of the components of these cases, the design of the knowledge-base and each of its tables was defined. Table 6.4.1.3 is an overview of key tables and their role in the knowledge base.

Table Name	Role/Information Held
FaultDetection_Cases	List of Cases for each Module Type. Each Case has a Unique ID, a Name and Description.
FaultDetection_Case_Notifications	Contains information on who to notify in case of a fault for each fault-case. Certain cases only apply for the Software Support team, others apply only to the Hardware Support team and some apply to both teams.
FaultDetection_Notification_Teams	The teams to notify. Although there are currently only 2 teams needed, in order to allow expandability, a specific table was designed.
FaultDetection_ParamTypes	The types of parameters that can be used for testing each case, these include: String, Dates, Integers, Module Attributes and Settings, Tests, Functions, etc.
FaultDetection_Params	List of all the available parameters (and their types) – parameters are kept in a separate table for • Database normalisation • The same parameter can be used in different fault cases tests (and test rules). See appendix A.1.5.1 for a complete list of Parameter Types
FaultDetection_Tests	Tests associated with each fault-case. The same test can be used more than once in different fault-cases.
FaultDetection_OperationTypes *(each has an implementation of the* ***IFaultDetectionOperationType*** *Interface described in section 6.3.3.2)*	After interviewing the domain experts, it became clear that there are different types of operations for different rules – for example: • IF X ***Contains*** Y THEN Passed • IF X ***Is_Longer_Than*** Y THEN Passed For database normalisation and expandability there are kept in a separate table. See appendix A.1.7.1 for a complete list of Operation Types
FaultDetection_Tests_Rules	The rules each test evaluates using one of the above Operation Types. A test can have unlimited number of rules to check and these can be joined with **AND**/**OR** relation.

Table 6.4.1.3 – Knowledge-Base Tables Overview

The complete structure of the knowledge-base and the tables is given in Appendix A.

6.4.2 Phase 2 - Foundations

The information collected in Phase 1 led to a conceptual idea of the system, which formed its foundation. With that, the following steps were taken to begin the development of the system.

6.4.2.1 Specification, documentation and generalisation:

Write up specification documents and testing documents – fault-cases, tests and rules were collected, grouped together and analysed to see which parameters were needed. An initial list of parameter types and operation types was also created.

6.4.2.2 First versions

Once enough information was gathered, and analysed, the first version of that knowledge-base was created including all the tables, views and stored-procedures (see Appendix A).

After the knowledge-base was created, the first version of the Fault Detection Engine Class Library (DLL) was created. Entity Classes (i.e. ***FaultCase***, ***FaultCaseCollection***, etc.), Enumerators (i.e. ***RelationType***, ***ParameterType***, etc.), the base structures (i.e. ***FaultDetectionResult***) and Interfaces were written. Once these were created an initial set of concrete classes implementing the interfaces was implemented (see Appendix B for full code listing).

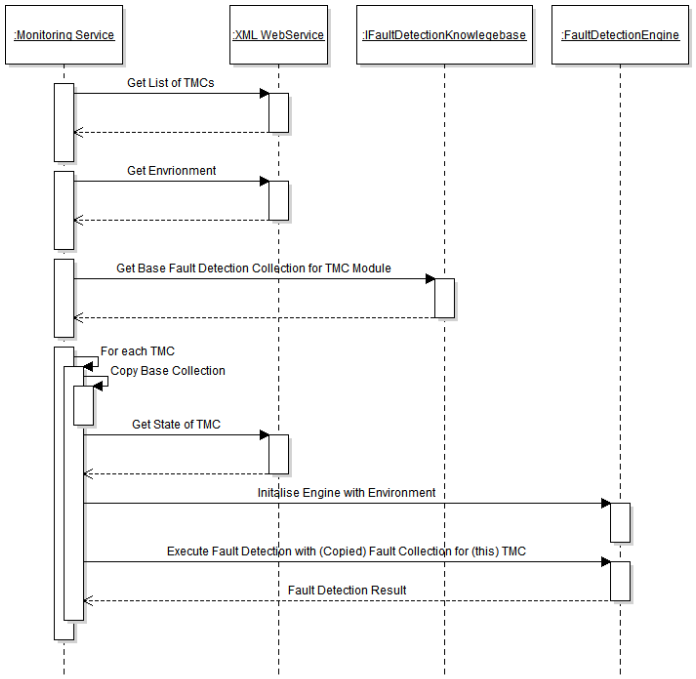


Figure 6.4.2.2.1 – Overview of a Single Fault Detection Iteration

When the initial version of the Fault Detection Engine DLL was completed, a Windows Service consuming the DLL was written, however, the development of this service only continued at a later

stage. Figure 6.4.2.2.1 details the basic sequence of actions of a single iteration of the service. It describes what the monitoring service will do on a configurable timed schedule.

The Tester App Approach

One of the key factors for the rapid development of this project was the concept of the Tester App (Figure 6.4.2.2.2). This is a very basic Windows Form application which loads the Fault Detection Engine DLL, and uses it exactly the same way the Windows Service (Figure 6.4.2.2.1) does and allows to quickly and efficiently test the inference engine and the rules by simulating the interaction with TMCs. Additionally, it allows to simulate fault-cases which are very hard to produce using the TMCs themselves, by manually editing the state JSON string.

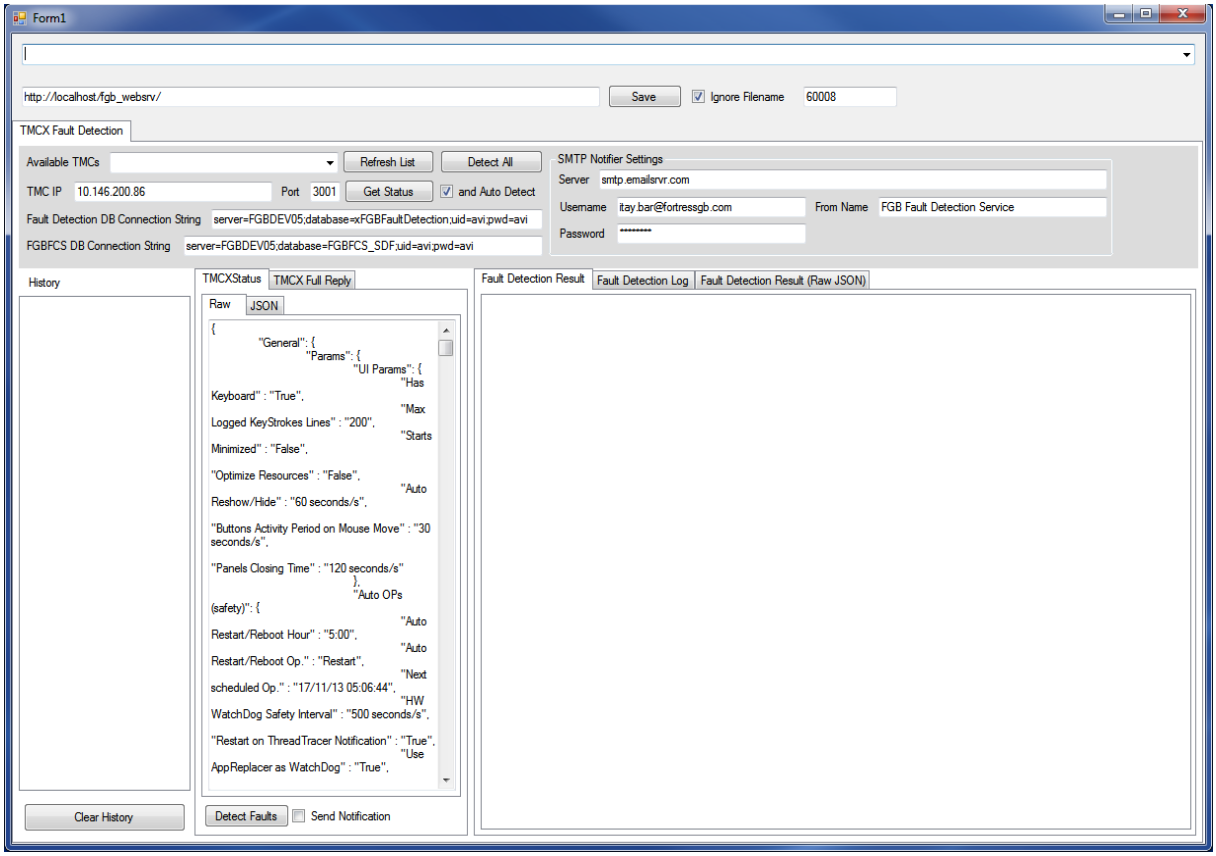


Figure 6.4.2.2.2– The Tester App

With these foundations in place, the development entered its 3rd phase.

6.4.3 Phase 3 – Evolutionary Development

Using the tester app, and the base Fault Detection Engine, the evolutionary development phase began. In this phase the Fault Detection Engine was examined and expanded, more rules were discovered, which led to new implementations of the Operation Type Interface.

It's important to note that in this stage, a new functionality was thought of, which would allow the Fault Detection the ability to also perform forward chaining, by creating Paths in the fault-cases tests, allowing the collection of "insights" from the TMC State String, however, following the MoSCoW prioritisation, it was decided that this feature will not be included in the scope of this project. This is explained further in section 8.

With a substantial set of fault-cases, tests and rules in place, the first implantation of the ***IFaultDetectionNotifier*** was created. This was a basic Email sending "notifier" which sends notifications to the *appropriate* support team once a Fault Detection Execution has ended with a fault detected. Figure 6.4.3.1 below described the expansion of a single fault detection iteration to include notifications and saving the execution results to the knowledge-base.

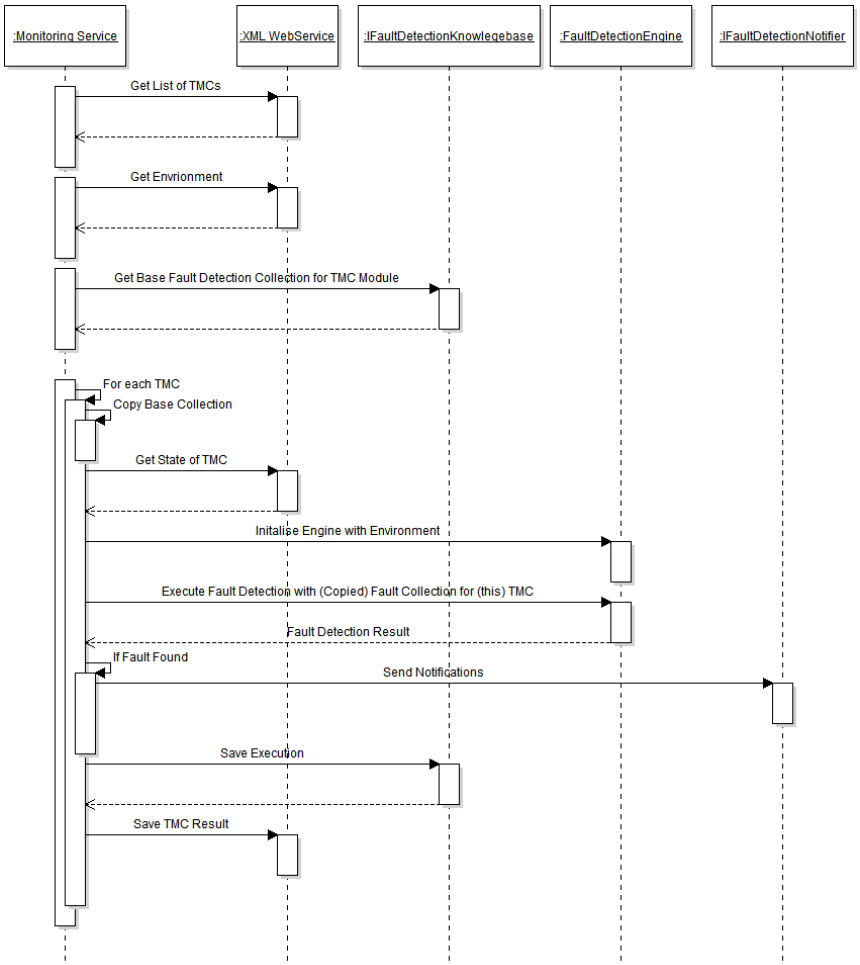


Figure 6.4.3.1 – Expanded Single Fault Detection Iteration, includes Notification sending and saving Execution Results

Using the Tester App, a large bank of "Past Executions" was created and saved to the knowledge-base, and with that data in place, the building of the API that provides that state of the TMCs and

the Web-Based Explanation facility started. This web-based interface (Figures 6.4.3.2 and 6.4.3.3), listing the TMCs and their current state, was developed quickly by using known JavaScript libraries such as jQuery and Knockout.js and tested extensively using the Firebug Firefox extension.

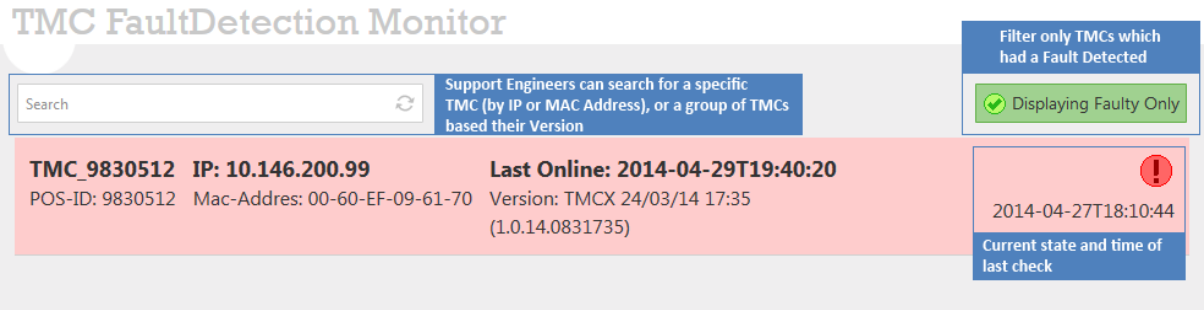


Figure 6.4.3.2 – The Web-Base Explanation Facility – Listing TMCs and their current state.

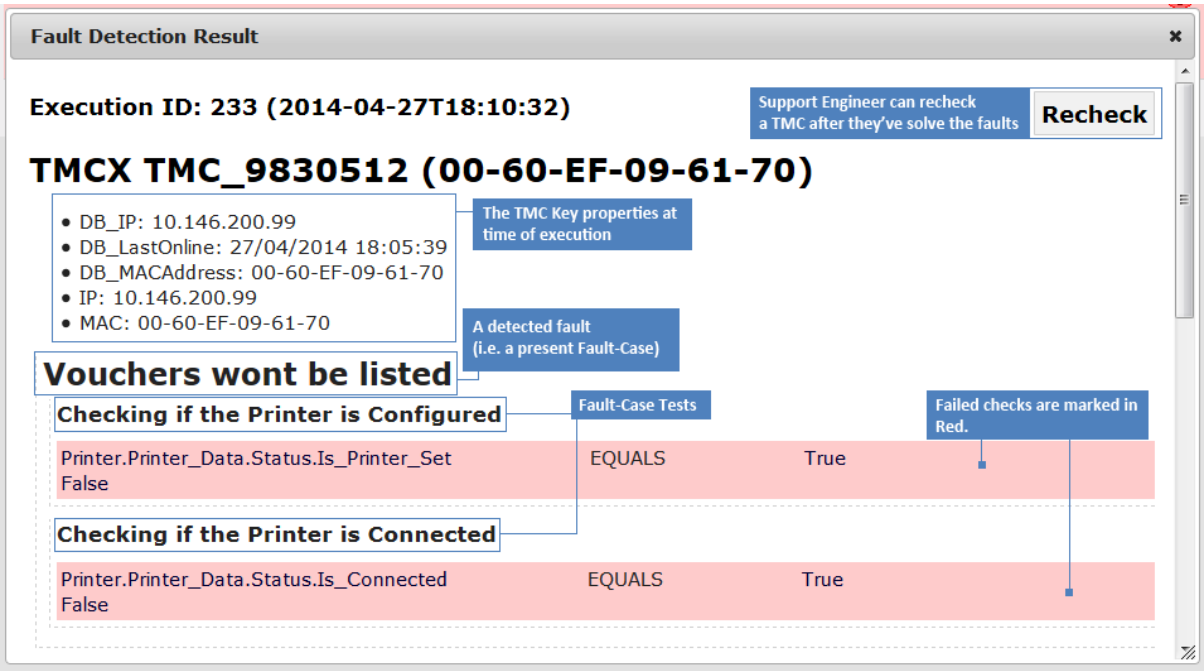


Figure 6.4.3.3 – The Web-Base Explanation Facility – Past Fault Detection Execution Explained.

With all the system components in place, the development of the Monitoring Service was also completed (using verified code taken from the Tester App) and the next phase started.

6.4.4 Phase 4 – Deployment

In this phase, the fine tuning of all the components in the system was done, using the Tester App initially, and then setting up dummy TMCs in the Fortress office to run full-loop tests. The support teams started experimenting with the explanation facility internally and provided feedback for further development.

The TMC developers also played a big role in validating their knowledge was being used correctly by the inference engine. This was iterative, back-and-forth process between the end-users (support teams), the domain experts (TMC developers), and the knowledge engineer and programmer.

Once all members were happy, all deliverables were then passed to the QA team which tested the solution according to requirements and testing documentation written up in Phase 2. The details of this activity are described in section 7.

7. Testing and Validation

7.1 Overview

In this project, testing was done throughout the development process, using the Tester App approach at the beginning, and then using a more structured scenario based testing. This process was iterative, and when issues were found and reported, a new build was created, which was tested again from the beginning (regression testing).

7.1.1 Scenario Based Testing

The method used for testing was “Scenario Based Testing”, the concept of which is to define or discover plausible scenarios and workflows that can imitate end user behaviour. For this approach to work, real-life domain knowledge is needed to create accurate and realistic scenarios (Crispin and Gregory, 2009)

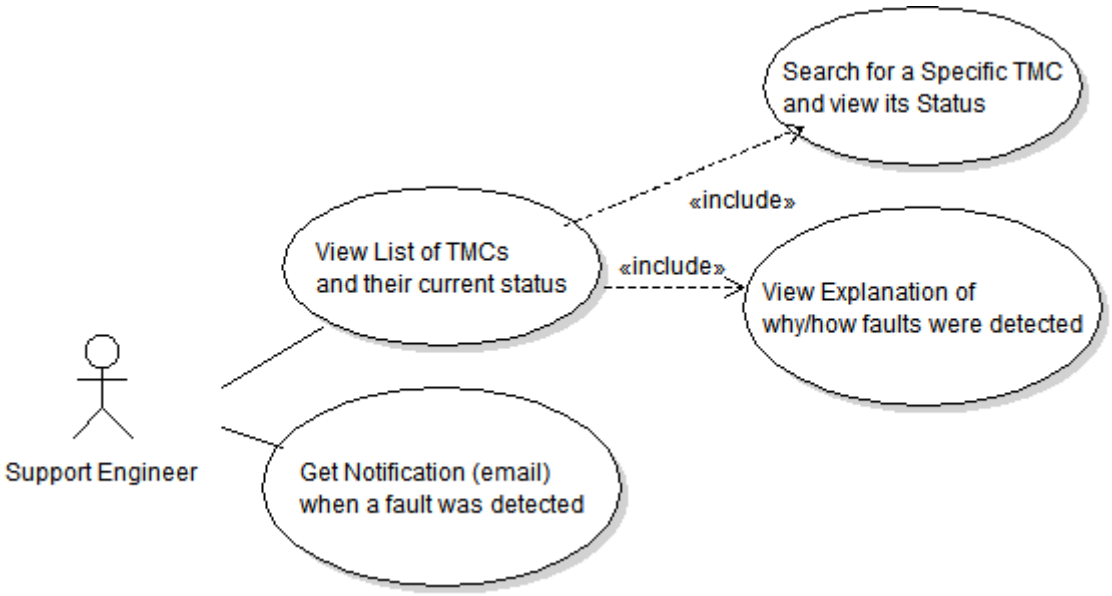


Figure 7.1.1.1 – Basic Support Engineer Use Cases

The starting point would be the basic use cases (Figure 7.1.1.1) of the software; however the idea is to go beyond those, since the aim of testing is to get the software to fail so errors could be found (Bennet et al, 2006).

The testing was split into 3 different categories each done at a separate phase of the development

7.2 Functional and Verification Testing

The verification and functional testing were focused more on scenarios related to "is the software behaving correctly?". These tests were split into two categories, Functional and Knowledge Verification and were performed by the developer using the Tester App.

7.2.1 Functional (The EngineTester Module)

For the purpose of functional testing, a duplicate of the TMC Module was implemented called EngineTester. The EngineTester module had its own fault-cases, tests and rules defined, covering all available Parameter Types and Operation Types, and all their possible combinations, similar to unit testing. Using the Tester App, a dummy JSON State String was loaded and parsed exactly the same way a TMC JSON State String would be, and then checked for faults. If a fault was detected, a bug in the Engine was found.

The EngineTester enabled end-to-end full cycle testing of all the engine functionalities and all its fault-cases and tests are detailed in Appendix C2.

7.2.2 Knowledge Verification

Once a functional deliverable was ready, the Tester App (including source code) and the Fault Detection Engine DLL (without the source code) were passed to the Domain Expert, to make sure that the Inference Engine was interpreting the state of the TMC correctly and preforming the checks accurately.

These tests were rapid, internal and undocumented. New builds of the DLL were created several times a day, every time an issue was discovered. Most of these tests were performed by generating a large set of TMC State Strings manually, each simulating different faults (that the domain experts intentionally created) and passed through the inference engine, then the results were manually verified.

7.3 Validation and Operational Testing

When the domain experts were happy with the results of the verification and functional testing, the Tester App (with its source code) and the DLL (without the source code) were passed to the second layer in which the testing scenarios more focused on “can the software actually be used?”

These scenarios were more real-life oriented, and were written up with the help of the support teams. The inner workings of the inference engine, or the rules in the knowledge-base were disregarded and looked at as “black box”. Real TMCs with real faults (for example, connecting a faulty printer or a faulty RFID reader) were used to create TMC State String and were passed to the Tester App, which invoked the Fault Detection Engine and reported the results. If an expected (known) fault was missed, or a false fault was detected, an issue was found.

See Appendix C for all the scenarios performed at this stage.

7.4 Beta (Pre-Production) Testing

This phase of testing is not part of the project proposal and will take place after the project deadline.
The monitoring service is to be installed (May, 2014) at one of Fortress American clients, which has over 300 TMCs in use.

The plan for this phase is to have the monitoring service work with real TMCs in a real environment, issuing notifications to the support teams. The idea is to see if any faults were missed by the service/fault detection engine and were reported by the club before a notification was sent. It will also give an opportunity to do load testing in a scale which is not possible to do manually.

Additionally, this phase will be used to gather more data and generate new knowledge (i.e. fault-cases and tests) in the knowledge-base.

8. Discussion and conclusions

8.1 Reflection

The original proposal for this project was to create a Fault Detection and Isolation (FDI) System for the TMC machines at Fortress. Throughout the development of this project, it became clear that if written and planned correctly, more can be accomplished.

One of the guidelines during the design of this solution was flexibility and expandability, which enabled to deliver a robust solution which can be adapted quickly, with minimal code changes to become an FDI system for other Fortress modules.

The choice of an extremely narrow domain (a software built in house) for an Expert System and the proximity and availability of the domain experts themselves proved to be very helpful. The work plan suggested in the proposal was flexible enough, and gave us enough time to prepare and build the solution.

The DSDM approach in general, and the time-boxing concept specifically gave a solid structure to the development process while enabling rapid progress, and agility and flexibility to adapt to changes in design quickly.

The MoSCoW prioritisation forced us to stay on track and not deviate from the plan at all, when a new feature or requirement came to light, if it didn't fall into the "Must Have" category it was discarded.

The Tester App Approach proved to be extremely helpful and reduced development time significantly. Debugging issues became a quicker and easier process which enabled rapid development without many delays.

8.2 Summary

The objectives of this project were to design and build an Artificial Intelligence Fault Detection and Isolation (FDI) software for a property software/hardware solution called TMC. It uses a rule-based knowledge-based system to detect faults and send notification to specific teams when needed and provide a web-based explanation facility for the end-users to view the current state of the TMCs being monitored.

The objectives have been met fully and on time, and the system is deployment ready and small scale production testing has begun.

8.3 Further Development

During the development of this project, a few additional features were proposed, however, due to time constrains and following the MoSCoW prioritisation (section 5) these were postponed for later date. Below is a list of some of the further development that will be taken at a future time:

8.3.1 Fault Cases Tests Paths

The idea behind adding Paths to fault case tests is that it will enable to navigate through the TMC state string dynamically, and will in essence give the inference engine the ability to also to use the forward chaining method. When a test is checked, it will have two paths defined, one to go to in case of failure, and one to go to in case of success. This will also enable to skip tests which are no longer relevant, once a preceding test has passed or failed, which will improve the efficiency of the inference engine greatly.

8.3.2 Fault Detection Parameter Type Interface

Changing the *FaultDetectionParamType* from an Enumerator to an Interface will enable creating new Parameter Types quickly and efficiently. This was not done for this project as the different parameter types of the TMC were clearly defined at the beginning and the need to expand them never came, however, if this solution is to be expanded and used on other Fortress modules, the possible parameter types to be used are endless.

8.3.4 Automated Knowledge Generation

Currently, the generation of rules in the knowledge-base is a manual procedure. As discussed in section 4, Feigenbaugm (1984) refers to this as the “bottleneck problem of AI”, which slows down the progression of this solution immensely.

During the development of this project, it came to light that there might be a way of generating rules in the knowledge-base by using regular-expressions to infer new rules from the TMC source code. The idea is to use Fortress Source Control software to detect changes in the TMC source code, and then use regular expression to scan the source code, detect special comments left by the developers and generate new rules in the knowledge-base. This would make the FDI solution proposed in this project completely self-sustained.

9. References

Bennet, S., McRobb S. and Farmer, R. 2006.Object Oriented Systems Analysis and Design using UML (3rd Edition). McGraw-Hill.

Chen, J. and Patton, R.J. 1999. Robust Model-based Fault Diagnosis for Dynamic Systems. Kluwer Academic Publishers.

Corsello, M. 2008. System-of-Systems Architectural Considerations for Complex Environments and Evolving Requirements, IEEE Systems Journal, Volume 2, Number 3, September 2008, pp. 312-320.

Craddock, A., Richards, K., Tudor, D., Roberts, B. and Godwin, J. 2012. The DSDM Agile Project Framework for Scrum [Online]. [Accessed 4 May 2014]. Available From: http://www.dsdm.org/sites/default/files/The%20DSDM%20Agile%20Project%20Framework%20v1%2011.pdf

Crispin, L. and Gregory, J. 2009. Agile Testing - A Practical Guide for Testers and Agile Teams. Addison-Wesley.

Ding, S. 2008. Model-based Fault Diagnosis Techniques – Design Schemes, Algorithms, and Tools. Springer-Verlag Berlin Heidelberg.

DSDM Consortium. 2014. Atern Principles [Online]. [Accessed 4 May 2014]. Available From: http://www.dsdm.org/content/4-atern-principles

DSDM Consortium. 2014. Introduction to the Handbook [Online]. [Accessed 4 May 2014]. Available From: http://www.dsdm.org/content/1-introduction-handbook

DSDM Consortium. 2014. The Lifecycle [Online]. [Accessed 4 May 2014]. Available From: http://www.dsdm.org/content/6-lifecycle

ECMA International. 2013. Standard ECMA-404 The JSON Data Interchange Format [Online]. [Accessed 4 May 2014]. Available From: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Feigenbaum, E. A. 1984. Knowledge Engineering The Applied Side of Artificial Intelligence, Annals of the New York Academy of Sciences 426, pp. 91-107.

Giarratano, J. and Riley G.D. 1998. Expert Systems Principles and Programming (3ed edition). Course Technology.

Isermann, R. 1997. Supervision, Fault-Detection and Fault-Diagnosis Methods - An Introduction, Control Engineering Practice, Volume 5, Issue 5, May 1997, pp. 639-652.

Isermann, R. and Balle, P. 1997. Trends in Application of Model-Based Fault Detection and Diagnosis of Technical Processes, Control Engineering Practice, Volume 5, Issue 5, May 1997, pp. 709-719.

James Newton-King. 2014. James Newton-King Json.NET [Online]. [Accessed 4 May 2014]. Available From: http://james.newtonking.com/json

Microsoft. 2014. Learn About ASP.NET Web API [Online]. [Accessed 4 May 2014]. Available From: http://www.asp.net/web-api

MSDN. 2005. An Introduction to C# Generics [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/ms379564%28v=vs.80%29.aspx

MSDN. 2014. Common Language Runtime (CLR) [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/8bs2ecf4%28v=vs.110%29.aspx

MSDN. 2014. Differences Between C++ Templates and C# Generics (C# Programming Guide) [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/c6cyy67b.aspx

MSDN. 2014. Getting Started with the .NET Framework [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/hh425099%28v=vs.110%29.aspx

MSDN. 2014. Introduction to the C# Language and the .NET Framework [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

MSDN. 2014. IXmlSerializable Interface [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/system.xml.serialization.ixmlserializable.aspx

MSDN. 2014. Microsoft SQL Server [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/bb545450%28v=sql.10%29.aspx

MSDN. 2014. namespace (C# Reference) [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/z2kcy19k.aspx

MSDN. 2014. Reflection in the .NET Framework [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/f7ykdhsy%28v=vs.110%29.aspx

MSDN. 2014. XML Web Services Overview [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/w9fdtx28%28v=aspnet.11%29.aspx

MSDN. 2014. .NET Framework Class Library Overview [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/hfa3fa08%28v=vs.110%29.aspx

MSDN. 2014. .NET Framework Regular Expressions [Online]. [Accessed 4 May 2014]. Available From: http://msdn.microsoft.com/en-us/library/hs600312%28v=vs.110%29.aspx

Negnevitsky, M. 2002. Artificial Intelligence: A Guide to Intelligent Systems (3rd Edition). Addison-Wesley.

Russell, S.J. and Norvig, P. 1995. Artificial Intelligence - A Modern Approach. Prentice-Hall, Inc.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, DC. 1996. Failure Diagnosis Using Discrete-Event Models, IEEE Transaction On Control Systems Technology, Volume 4, Issue 2, March 1996, pp. 105-124.

Sedighi, T., Phillips, P., Foote, P. 2013. Model-based intermittent fault detection, Procedia CIRP, Volume 11, 2013, pp. 68-73.

W3Schools.com. 2014. JSON Tutorial [Online]. [Accessed 4 May 2014]. Available From: http://www.w3schools.com/json/default.asp

10. Appendixes

Appendix A – Knowledge-Base

1. Database Tables

1.1 Entity Relationship Diagram

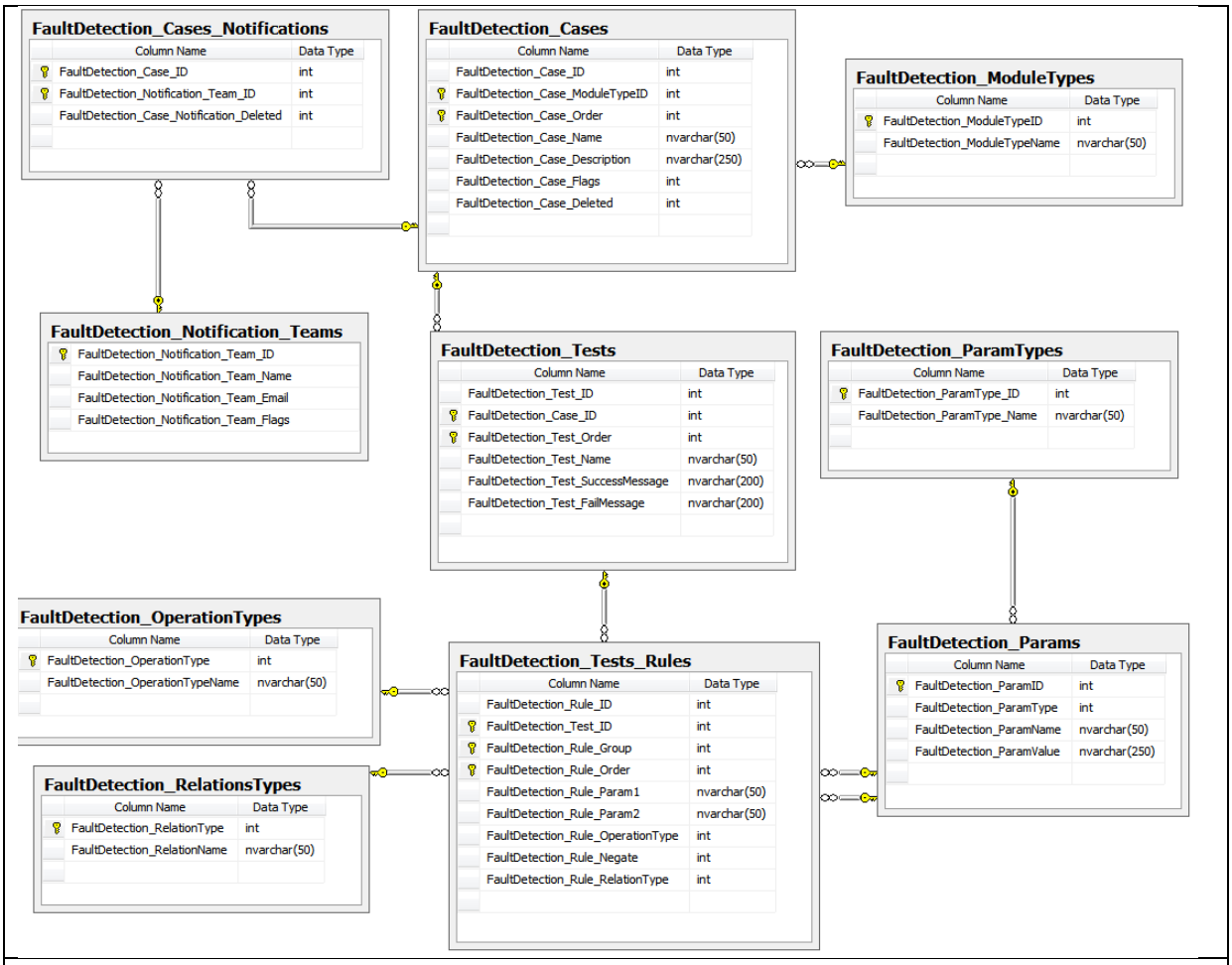


Figure A1.1.1 – Knowledge-Base Entity Relationship Diagram – Directed Export from SSMS.

1.2 FaultDetection_Cases

Holds the different fault cases and their execution order, for each module type.

Column Name	Notes	Description
FaultDetection_Case_ID	Unique ID	The unique ID of the Case.
FaultDetection_Case_ModuleTypeID	FK to FaultDetection_ModuleTypes Composite PK	The ModuleType the case belongs to.
FaultDetection_Case_Order	Composite PK	The place/order, i.e. when to be executed.
FaultDetection_Case_Name		The Case Name
FaultDetection_Case_Description		The Case Description
FaultDetection_Case_Flags		The Case Bitwise Flags: 0 - NONE 1 - IGNORE_FAILED 2 - NO_NOTIFICATIONS
FaultDetection_Case_Deleted		1 – Deleted, 0 – Not Deleted.

1.3 FaultDetection_Tests

Holds the Fault Cases Tests, and their execution order for each Case.

Column Name	Notes	Description
FaultDetection_Test_ID	Unique Test ID	The unique ID of the Test
FaultDetection_Case_ID	FK to FaultDetection_Cases Composite PK	The case the test belongs to.
FaultDetection_Test_Order	Composite PK	The place/order, i.e. when to be executed.
FaultDetection_Test_Name		The Tests Name
FaultDetection_Test_SuccessMessage		A Message to display when test passes
FaultDetection_Test_FailMessage		A Message to display when test fails.

1.4 FaultDetection_Tests_Rules

Holds the different rules their groups and order, and the parameters they evaluate for each fault case test.

Column Name	Notes	Description
FaultDetection_Rule_ID	Unique Rule ID	The unique ID of the rule
FaultDetection_Test_ID	FK to FaultDetection_Tests Composite PK	The test the rule belongs to
FaultDetection_Rule_Group	Composite PK	The group (in the test) the rule belongs to
FaultDetection_Rule_Order	Composite PK	The place/order of the rule in the group

FaultDetection_Rule_Param1	FK to FaultDetection_Params	The 1st parameter to evaluate
FaultDetection_Rule_Param2	FK to FaultDetection_Params	The 2nd parameter to evaluate
FaultDetection_Rule_OperationType	FK to FaultDetection_OperationTypes	The operation to use in evaluation
FaultDetection_Rule_Negate		1 to Negate to result (equivalent to ! or NOT, i.e. if operation is EQUALS, negate means NOT EQUALS)
FaultDetection_Rule_RelationType	FK to FaultDetection_RelationsTyes	The relationship (AND/OR) to the next rule or group.

1.5 FaultDetection_ParamTypes

Holds the different parameter types and their names.

Column Name	Notes	Description
FaultDetection_ParamType_ID	PK, Unique ID	The unique ID of the parameter type
FaultDetection_ParamType_Name	Unique Key	The unique name of the parameter type

1.5.1 Supported Parameter Types

Type ID	Type Name	Notes
0	NULL	Represents null
1	STRING	Represents string or String
2	INTEGER	Represents int or Int32
3	DECIMAL	Represents decimal or Decimal
4	STRING_ARRAY	Represents string[] or String[]
5	INT_ARRAY	Represents int[] or Int32[]
6	DECIMAL_ARRAY	Represents decimal[] or Decimal
7	BOOLEAN	Represents bool or Boolean
10	DATE	Represents DateTime
11	TIMESPAN	Represents TimeSpan
20	MODULE_ATTRIBUTE	Represents a Module State Attribute name.
21	MODULE_SETTING	Represents a Module Settings name
40	TEST_ID	Represents a link to another Test (by its ID)
50	ENVIRONMENT_ATTRIBUTE	Represents an Environment Attribute name.
60	REGEX_PATTERN	A Regular Expressions Pattern
70	FUNCTION	A Function to execute.

1.6 FaultDetection_Params

Holds all the parameters, their types, names and values. These are held in a different table since each parameter can be used multiple times (in different rules, test and cases).

Column Name	Notes	Description
FaultDetection_ParamID	PK, Unique ID	The unique ID of the parameter
FaultDetection_ParamType	FK to FaultDetection_ParamTypes	The type of the parameter
FaultDetection_ParamName	Unique Key	The parameter name
FaultDetection_ParamValue		The parameter value – depends on the parameter type.

1.7 FaultDetection_OpertionTypes

Holds all the operation types and their names.

Column Name	Notes	Description
FaultDetection_OperationType	PK, Unique ID	The unique id of the operation type
FaultDetection_OperationTypeName	Unique Key	The operation type name

1.7.1 Supported Operations and their acceptable Parameter Types Combinations

Operation Type Name	Supported Parameter Types Combinations
CONTAINS	• STRING, STRING • STRING_ARRAY, STRING • STRING_ARRAY, REGEX_PATTERN • INT_ARRAY, INTEGER • DECIMAL_ARRAY, DECIMAL
EQUALS	• NULL, NULL • STRING, STRING • STRING, REGEX_PATTERN • REGEX_PATTERN, STRING • INTEGER, INTEGER • INTEGER, DECIMAL • DECIMAL, DECIMAL • DECIMAL, INTEGER • BOOLEAN, BOOLEAN • INT_ARRAY, INT_ARRAY • STRING_ARRAY, STRING_ARRAY • DECIMAL_ARRAY, DECIMAL_ARRAY • DATE, DATE • TIMESPAN, TIMESPAN
GREATER_EQUALS	• INTEGER, INTEGER • INTEGER, DECIMAL • DECIMAL, DECIMAL • DECINAL, INTEGER • DATE, DATE
GREATER_THAN	• INTEGER, INTEGER • INTEGER, DECIMAL

	• DECIMAL, DECIMAL • DECINAL, INTEGER • DATE, DATE
LESS_EQUALS	• INTEGER, INTEGER • INTEGER, DECIMAL • DECIMAL, DECIMAL • DECINAL, INTEGER • DATE, DATE
LESS_THAN	• INTEGER, INTEGER • INTEGER, DECIMAL • DECIMAL, DECIMAL • DECINAL, INTEGER • DATE, DATE
LONGER_EQUALS	• STRING, STRING • STRING_ARRAY, STRING_ARRAY • INT_ARRAY, INT_ARRAY • DECIMAL_ARRAY, DECIMAL_ARRAY • TIMESPAN, TIMESPAN
LONGER_THAN	• STRING, STRING • STRING_ARRAY, STRING_ARRAY • INT_ARRAY, INT_ARRAY • DECIMAL_ARRAY, DECIMAL_ARRAY • TIMESPAN, TIMESPAN
SHORTER_EQUALS	• STRING, STRING • STRING_ARRAY, STRING_ARRAY • INT_ARRAY, INT_ARRAY • DECIMAL_ARRAY, DECIMAL_ARRAY • TIMESPAN, TIMESPAN
SHORTER_THAN	• STRING, STRING • STRING_ARRAY, STRING_ARRAY • INT_ARRAY, INT_ARRAY • DECIMAL_ARRAY, DECIMAL_ARRAY • TIMESPAN, TIMESPAN

1.8 FaultDetection_RelationsTypes

Holds the different relationships between single test rules or their groups. (AND, OR and EOF – END OF FILE)

Column Name	Notes	Description
FaultDetection_RelationType	PK, Unique ID	The unique id of the relation type
FaultDetection_RelationName	Unique Key	The relation type name (EOF, AND, OR)

1.9 FaultDetection_ModuleTypes

Holds the different module types and their names.

Column Name	Notes	Description
FaultDetection_ModuleTypeID	PK, Unique ID	The unique id of the module type
FaultDetection_ModuleTypeName	Unique Key	The name of the module type

1.10 FaultDetection_Case_Notifications

Holds the notification team registration information for each fault case.

Column Name	Notes	Description
FaultDetection_Case_ID	FK to FaultDetection_Cases Composite PK	The ID of the Case this notification registration is for.
FaultDetection_Notification_Team_ID	FK to FaultDetection_Notitification_Teams Composite PK	The ID of the team this registrations is for.
FaultDetection_Case_Notification_Deleted		0 – registration active, 1 – registration deleted.

1.11 FaultDetection_Notification_Teams

Holds the information for each notification team.

Column Name	Notes	Description
FaultDetection_Notification_Team_ID	PK, Unique ID	The ID of the notification team
FaultDetection_Notification_Team_Name		The team name
FaultDetection_Notification_Team_Email		The email address to send notifications to.
FaultDetection_Notification_Team_Flags		Bitwise flags (for future use)

1.12 Past Executions Tables

1.12.1 Past Executions Entity Relationship Diagram

Figure A1.12.1.1 – Knowledge-Base Past Executions Entity Relationship Diagram – Directed Export from SSMS.

1.12.2 FaultDetection_Executions

Holds information about past fault detection executions.

Column Name	Notes	Description
FaultDetection_Execution_ID	PK, Unique ID	The ID of the past execution. Gets attached to the TMCs Table on the Fortress Database.
FaultDetection_Execution_Module_TypeID		The type id of the module the execution was on.
FaultDetection_Execution_Module_UniqueId		The unique identifier of the module the execution was on. *In TMCs the unique identifier is their MAC Address.*
FaultDetection_Execution_Module_UniqueName		The unique name of the module the execution was on. *In TMCs this is their Till Name.*
FaultDetection_Execution_Time		The time of execution

FaultDetection_Execution_Duration		The duration (how long) it took to perform the check.
FaultDetection_Execution_Result		The result of the execution
FaultDetection_Execution_Error		Description of the failed result (if applicable)
FaultDetection_Execution_LastUpdate		The Last Update time when this record was updated.
FaultDetection_Execution_Deleted		1 – Deleted, 0 – Not Deleted.

1.12.3 FaultDetection_Executions_ModuleProperties

Holds different module properties and values at the time of a fault detection execution.

Column Name	Notes	Description
FaultDetection_Execution_ID	FK to FaultDetection_Executions	The ID of the execution this property belongs to.
FaultDetection_Module_Property_Name		The property name
FaultDetection_Module_Property_Value		The property value at time of execution.
FaultDetection_Module_Property_LastUpdate		The Last Update time when this record was updated.

1.12.4 FaultDetection_Executions_Cases

Holds all the fault cases and their results for each past fault detection execution.

Column Name	Notes	Description
FaultDetection_Execution_ID	FK to FaultDetection_Executions Composite PK	The ID of the execution this case belongs to.
FaultDetection_Execution_Case_ID	Composite PK	The Case ID
FaultDetection_Execution_Case_Result		The result of the case execution.
FaultDetection_Execution_Case_Error		The description of faults (if applicable) found in the execution.
FaultDetection_Execution_Case_LastUpdate		The Last Update time when this record was updated.

1.12.5 FaultDetection_Executions_Tests

Holds all the fault case tests and their results for each past fault detection execution.

Column Name	Notes	Description
FaultDetection_Execution_ID	FK to FaultDetection_Executions Composite PK	The ID of the execution this case belongs to.
FaultDetection_Execution_Case_ID	Composite PK	The Case ID this test belongs to
FaultDetection_Execution_Test_Order	Composite PK	The Order of the test (in the case)

FaultDetection_Execution_Test_ID	Composite PK	The ID of the test
FaultDetection_Execution_Test_Result		The result of the test execution.
FaultDetection_Execution_Test_Error		The description of faults (if applicable) found in the execution.
FaultDetection_Execution_Test_LastUpdate		The Last Update time when this record was updated.

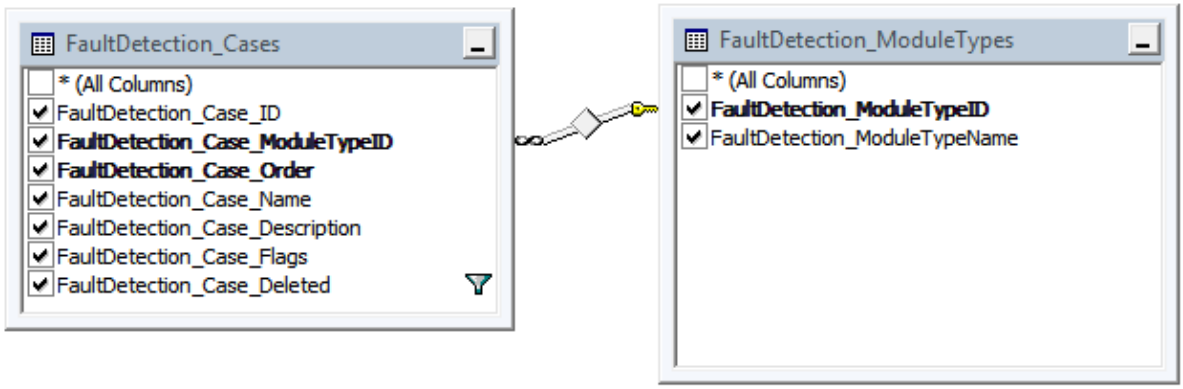
1.12.5 FaultDetection_Executions_Tests_Rules

Holds all the fault case test rules and their results and parameter computed values at time of execution.

Column Name	Notes	Description
FaultDetection_Execution_ID	FK to FaultDetection_Executions Composite PK	The ID of the execution this case belongs to.
FaultDetection_Execution_Test_RuleID	Composite PK	The Rule ID
FaultDetection_Execution_Test_ID		The ID of the test this rule belongs to
FaultDetection_Execution_Param1_ComputedType		Parameter 1 Computed Type at time of execution
FaultDetection_Execution_Param1_ComputedValue		Parameter 1 Computed Value at time of execution
FaultDetection_Execution_Param2_ComputedType		Parameter 2 Computed Type at time of execution
FaultDetection_Execution_Param2_ComputedValue		Parameter 2 Computed Value at time of execution
FaultDetection_Execution_Test_Rule_Result		The result of the test rule execution.
FaultDetection_Execution_Test_Rule_Error		The description of faults (if applicable) found in the execution.
FaultDetection_Execution_Test_Rule_LastUpdate		The Last Update time when this record was updated.

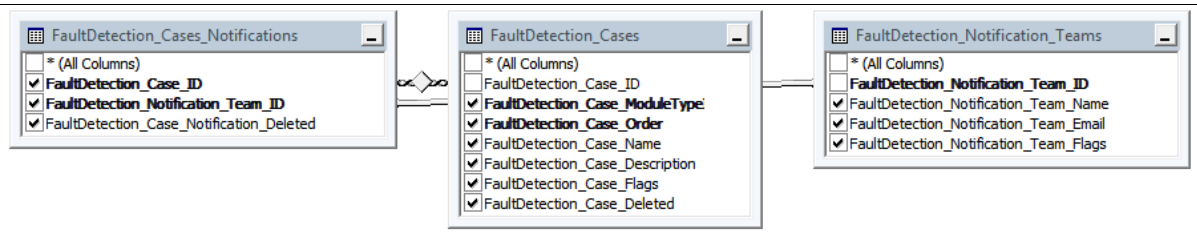
2. Database Views

2.1 vFaultDetection_Cases




```
SELECT     dbo.FaultDetection_Cases.FaultDetection_Case_ID,
dbo.FaultDetection_Cases.FaultDetection_Case_ModuleTypeID,
                dbo.FaultDetection_Cases.FaultDetection_Case_Order,
dbo.FaultDetection_Cases.FaultDetection_Case_Name,
                dbo.FaultDetection_Cases.FaultDetection_Case_Description,
dbo.FaultDetection_Cases.FaultDetection_Case_Flags,
                dbo.FaultDetection_Cases.FaultDetection_Case_Deleted,
dbo.FaultDetection_ModuleTypes.FaultDetection_ModuleTypeID,
                dbo.FaultDetection_ModuleTypes.FaultDetection_ModuleTypeName
FROM       dbo.FaultDetection_Cases INNER JOIN
                dbo.FaultDetection_ModuleTypes ON
                dbo.FaultDetection_Cases.FaultDetection_Case_ModuleTypeID =
dbo.FaultDetection_ModuleTypes.FaultDetection_ModuleTypeID
WHERE     (dbo.FaultDetection_Cases.FaultDetection_Case_Deleted = 0)
```

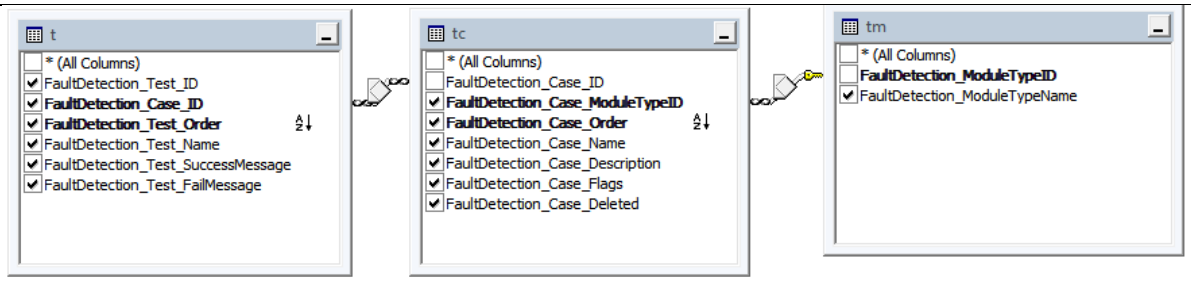
2.2 vFaultDetection_Cases_Notifications




```
SELECT     dbo.FaultDetection_Cases_Notifications.FaultDetection_Case_ID,
dbo.FaultDetection_Cases_Notifications.FaultDetection_Notification_Team_ID,
                dbo.FaultDetection_Cases_Notifications.FaultDetection_Case_Notification_Deleted,
dbo.FaultDetection_Cases.FaultDetection_Case_ModuleTypeID,
                dbo.FaultDetection_Cases.FaultDetection_Case_Order,
dbo.FaultDetection_Cases.FaultDetection_Case_Name,
                dbo.FaultDetection_Cases.FaultDetection_Case_Description,
dbo.FaultDetection_Cases.FaultDetection_Case_Flags,
                dbo.FaultDetection_Cases.FaultDetection_Case_Deleted,
dbo.FaultDetection_Notification_Teams.FaultDetection_Notification_Team_Name,
                dbo.FaultDetection_Notification_Teams.FaultDetection_Notification_Team_Email,
                dbo.FaultDetection_Notification_Teams.FaultDetection_Notification_Team_Flags
FROM       dbo.FaultDetection_Cases_Notifications INNER JOIN
                dbo.FaultDetection_Cases ON
dbo.FaultDetection_Cases_Notifications.FaultDetection_Case_ID =
```

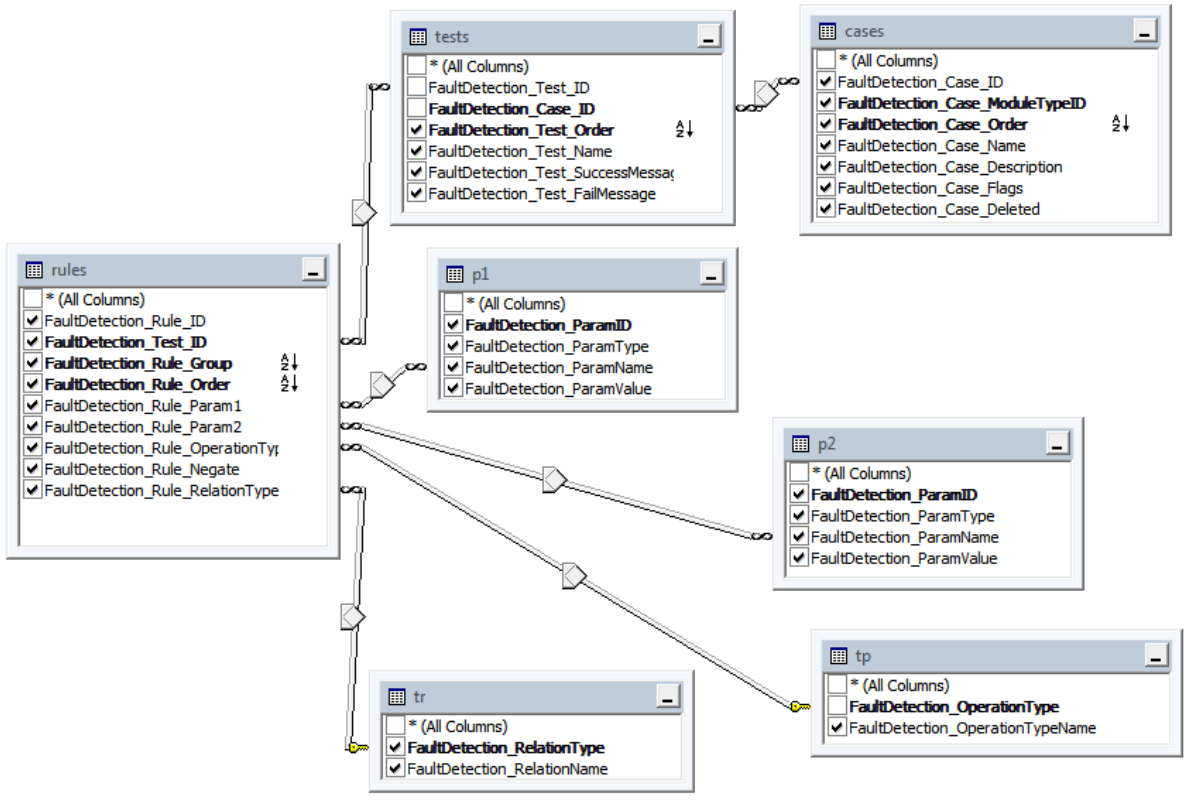
```
dbo.FaultDetection_Cases.FaultDetection_Case_ID INNER JOIN
                dbo.FaultDetection_Notification_Teams ON
                dbo.FaultDetection_Cases_Notifications.FaultDetection_Notification_Team_ID =
dbo.FaultDetection_Notification_Teams.FaultDetection_Notification_Team_ID
```

2.3 vFaultDetection_Tests




```
SELECT     TOP (100) PERCENT t.FaultDetection_Test_ID, t.FaultDetection_Case_ID,
t.FaultDetection_Test_Order, t.FaultDetection_Test_Name,
                t.FaultDetection_Test_SuccessMessage, t.FaultDetection_Test_FailMessage,
tc.FaultDetection_Case_ModuleTypeID, tc.FaultDetection_Case_Order,
                tc.FaultDetection_Case_Name, tc.FaultDetection_Case_Description,
tc.FaultDetection_Case_Flags, tc.FaultDetection_Case_Deleted,
                tm.FaultDetection_ModuleTypeName
FROM        dbo.FaultDetection_Tests AS t LEFT OUTER JOIN
                dbo.FaultDetection_Cases AS tc ON tc.FaultDetection_Case_ID =
t.FaultDetection_Case_ID LEFT OUTER JOIN
                dbo.FaultDetection_ModuleTypes AS tm ON tm.FaultDetection_ModuleTypeID =
tc.FaultDetection_Case_ModuleTypeID
ORDER BY tc.FaultDetection_Case_Order, t.FaultDetection_Test_Order
```

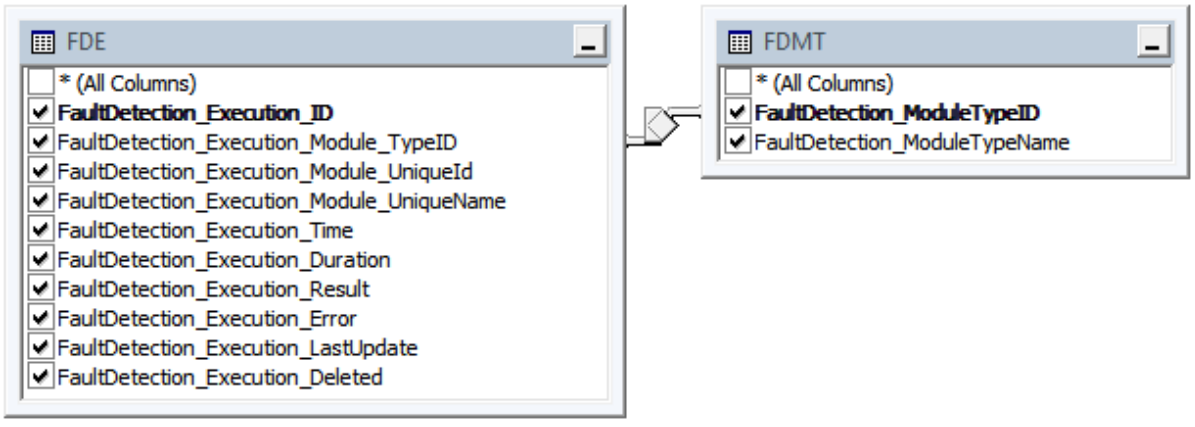
2.4 vFaultDetection_Tests_Rules



```
SELECT     TOP (100) PERCENT rules.FaultDetection_Rule_ID, rules.FaultDetection_Test_ID,
rules.FaultDetection_Rule_Group, rules.FaultDetection_Rule_Order,
                      rules.FaultDetection_Rule_Param1, rules.FaultDetection_Rule_Param2,
rules.FaultDetection_Rule_OperationType, rules.FaultDetection_Rule_Negate,
                      rules.FaultDetection_Rule_RelationType, tp.FaultDetection_OperationTypeName,
tr.FaultDetection_RelationName,
                      p1.FaultDetection_ParamType AS FaultDetection_Rule_Param1_Type,
p1.FaultDetection_ParamValue AS FaultDetection_Rule_Param1_Value,
                      p1.FaultDetection_ParamID AS FaultDetection_Rule_Param1_ID,
p1.FaultDetection_ParamName AS FaultDetection_Rule_Param1_Name,
                      p2.FaultDetection_ParamType AS FaultDetection_Rule_Param2_Type,
p2.FaultDetection_ParamValue AS FaultDetection_Rule_Param2_Value,
                      p2.FaultDetection_ParamID AS FaultDetection_Rule_Param2_ID,
p2.FaultDetection_ParamName AS FaultDetection_Rule_Param2_Name,
                      tests.FaultDetection_Test_Order, tests.FaultDetection_Test_Name,
tests.FaultDetection_Test_SuccessMessage, tests.FaultDetection_Test_FailMessage,
                      cases.FaultDetection_Case_ID, cases.FaultDetection_Case_ModuleTypeID,
cases.FaultDetection_Case_Order, cases.FaultDetection_Case_Name,
                      cases.FaultDetection_Case_Description, cases.FaultDetection_Case_Flags,
cases.FaultDetection_Case_Deleted, tr.FaultDetection_RelationType
FROM       dbo.FaultDetection_Tests_Rules AS rules LEFT OUTER JOIN
                      dbo.FaultDetection_Params AS p1 ON p1.FaultDetection_ParamName =
rules.FaultDetection_Rule_Param1 LEFT OUTER JOIN
                      dbo.FaultDetection_Params AS p2 ON p2.FaultDetection_ParamName =
rules.FaultDetection_Rule_Param2 LEFT OUTER JOIN
                      dbo.FaultDetection_OperationTypes AS tp ON tp.FaultDetection_OperationType =
rules.FaultDetection_Rule_OperationType LEFT OUTER JOIN
                      dbo.FaultDetection_RelationsTypes AS tr ON tr.FaultDetection_RelationType =
rules.FaultDetection_Rule_RelationType LEFT OUTER JOIN
                      dbo.FaultDetection_Tests AS tests ON tests.FaultDetection_Test_ID =
rules.FaultDetection_Test_ID LEFT OUTER JOIN
                      dbo.FaultDetection_Cases AS cases ON cases.FaultDetection_Case_ID =
tests.FaultDetection_Case_ID
ORDER BY cases.FaultDetection_Case_Order, tests.FaultDetection_Test_Order,
rules.FaultDetection_Rule_Group, rules.FaultDetection_Rule_Order
```

2.5 Past Executions Views

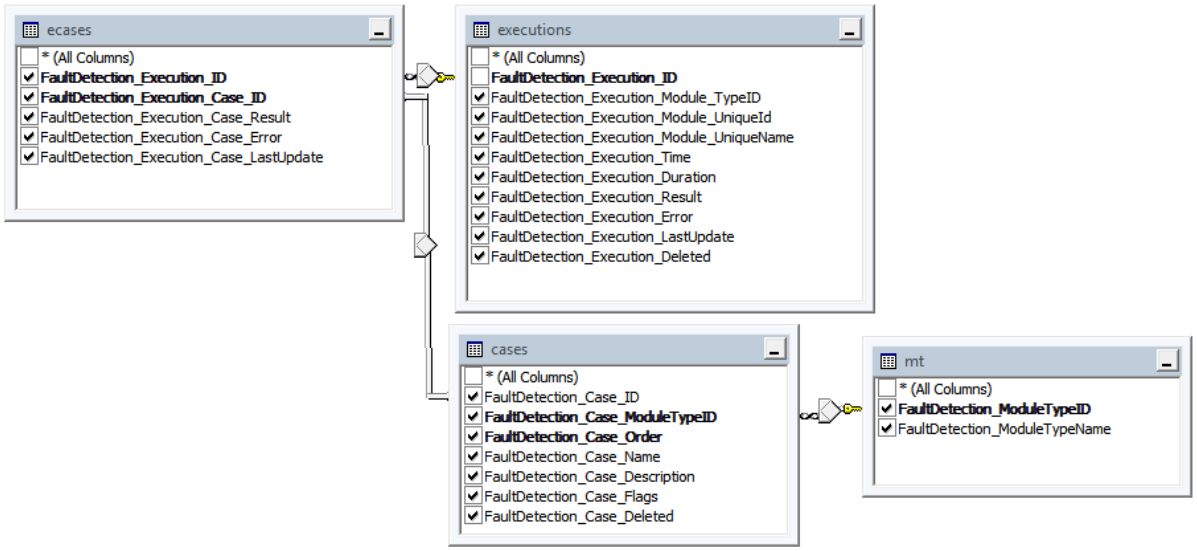
2.5.1 vFaultDetection_Executions




```
SELECT     FDE.FaultDetection_Execution_ID, FDE.FaultDetection_Execution_Module_TypeID,
```

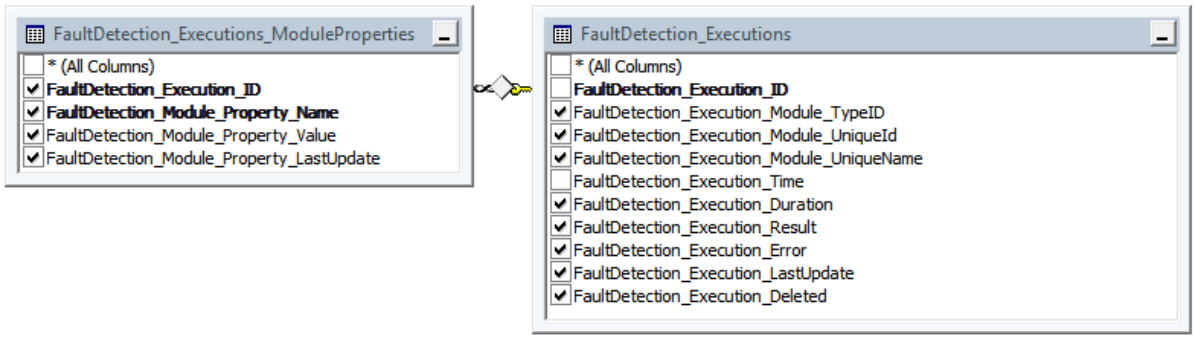
```
FDE.FaultDetection_Execution_Module_UniqueId,
            FDE.FaultDetection_Execution_Module_UniqueName,
FDE.FaultDetection_Execution_Time, FDE.FaultDetection_Execution_Duration,
            FDE.FaultDetection_Execution_Result, FDE.FaultDetection_Execution_Error,
FDE.FaultDetection_Execution_LastUpdate, FDE.FaultDetection_Execution_Deleted,
            FDMT.FaultDetection_ModuleTypeID, FDMT.FaultDetection_ModuleTypeName
FROM        dbo.FaultDetection_Executions AS FDE LEFT OUTER JOIN
            dbo.FaultDetection_ModuleTypes AS FDMT ON FDMT.FaultDetection_ModuleTypeID =
FDE.FaultDetection_Execution_Module_TypeID
```

2.5.2 vFaultDetection_Executions_Cases



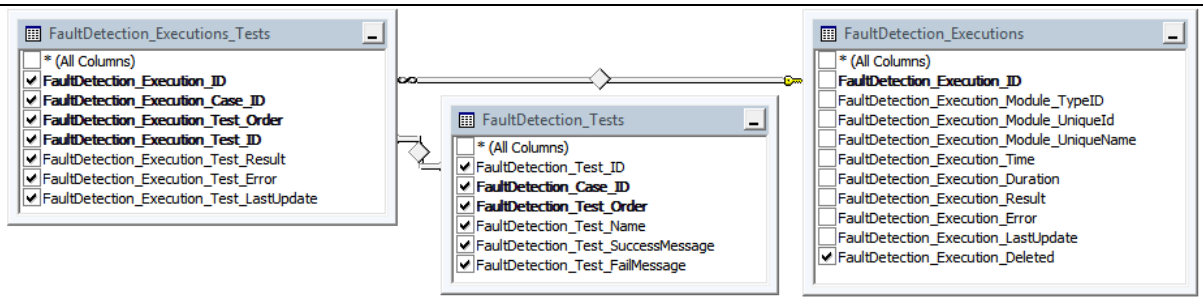

```
SELECT    ecases.FaultDetection_Execution_ID, ecases.FaultDetection_Execution_Case_ID,
ecases.FaultDetection_Execution_Case_Result,
            ecases.FaultDetection_Execution_Case_Error,
ecases.FaultDetection_Execution_Case_LastUpdate, cases.FaultDetection_Case_ID,
            cases.FaultDetection_Case_ModuleTypeID, cases.FaultDetection_Case_Order,
cases.FaultDetection_Case_Name, cases.FaultDetection_Case_Description,
            cases.FaultDetection_Case_Flags, cases.FaultDetection_Case_Deleted,
executions.FaultDetection_Execution_Module_TypeID,
            executions.FaultDetection_Execution_Module_UniqueId,
executions.FaultDetection_Execution_Module_UniqueName,
executions.FaultDetection_Execution_Time,
            executions.FaultDetection_Execution_Duration,
executions.FaultDetection_Execution_Result, executions.FaultDetection_Execution_Error,
            executions.FaultDetection_Execution_LastUpdate,
executions.FaultDetection_Execution_Deleted, mt.FaultDetection_ModuleTypeID,
            mt.FaultDetection_ModuleTypeName
FROM        dbo.FaultDetection_Executions_Cases AS ecases LEFT OUTER JOIN
            dbo.FaultDetection_Executions AS executions ON
executions.FaultDetection_Execution_ID = ecases.FaultDetection_Execution_ID LEFT OUTER JOIN
            dbo.FaultDetection_Cases AS cases ON ecases.FaultDetection_Execution_Case_ID =
cases.FaultDetection_Case_ID LEFT OUTER JOIN
            dbo.FaultDetection_ModuleTypes AS mt ON mt.FaultDetection_ModuleTypeID =
cases.FaultDetection_Case_ModuleTypeID
```

2.5.3 vFaultDetection_Executions_ModuleProperties



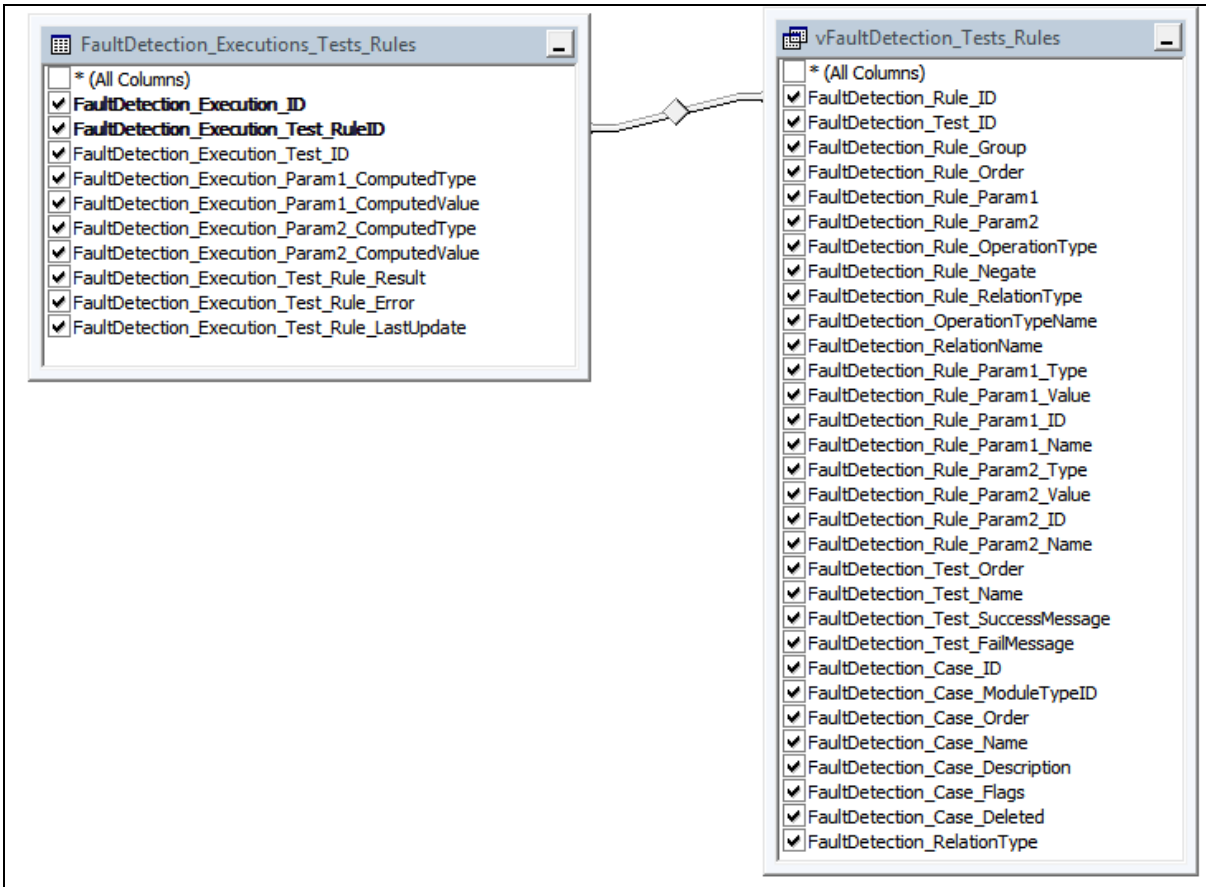

```
SELECT     dbo.FaultDetection_Executions_ModuleProperties.FaultDetection_Execution_ID,

dbo.FaultDetection_Executions_ModuleProperties.FaultDetection_Module_Property_Name,

dbo.FaultDetection_Executions_ModuleProperties.FaultDetection_Module_Property_Value,

dbo.FaultDetection_Executions_ModuleProperties.FaultDetection_Module_Property_LastUpdate,
                dbo.FaultDetection_Executions.FaultDetection_Execution_Module_TypeID,
dbo.FaultDetection_Executions.FaultDetection_Execution_Module_UniqueId,
                dbo.FaultDetection_Executions.FaultDetection_Execution_Module_UniqueName,
dbo.FaultDetection_Executions.FaultDetection_Execution_Duration,
                dbo.FaultDetection_Executions.FaultDetection_Execution_Result,
dbo.FaultDetection_Executions.FaultDetection_Execution_Error,
                dbo.FaultDetection_Executions.FaultDetection_Execution_LastUpdate,
dbo.FaultDetection_Executions.FaultDetection_Execution_Deleted
FROM        dbo.FaultDetection_Executions_ModuleProperties INNER JOIN
                dbo.FaultDetection_Executions ON
                dbo.FaultDetection_Executions_ModuleProperties.FaultDetection_Execution_ID =
dbo.FaultDetection_Executions.FaultDetection_Execution_ID
```

2.5.4 vFaultDetection_Executions_Tests




```
SELECT     dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_ID,
dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_Case_ID,
                dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_Test_Order,
dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_Test_ID,
                dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_Test_Result,
dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_Test_Error,
                dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_Test_LastUpdate,
dbo.FaultDetection_Tests.FaultDetection_Test_ID,
                dbo.FaultDetection_Tests.FaultDetection_Case_ID,
dbo.FaultDetection_Tests.FaultDetection_Test_Order,
dbo.FaultDetection_Tests.FaultDetection_Test_Name,
                dbo.FaultDetection_Tests.FaultDetection_Test_SuccessMessage,
dbo.FaultDetection_Tests.FaultDetection_Test_FailMessage,
                dbo.FaultDetection_Executions.FaultDetection_Execution_Deleted
FROM        dbo.FaultDetection_Executions_Tests INNER JOIN
                dbo.FaultDetection_Tests ON
dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_Test_ID =
dbo.FaultDetection_Tests.FaultDetection_Test_ID INNER JOIN
                dbo.FaultDetection_Executions ON
dbo.FaultDetection_Executions_Tests.FaultDetection_Execution_ID =
dbo.FaultDetection_Executions.FaultDetection_Execution_ID
```

2.5.5 vFaultDetection_Executions_Tests_Rules




```
SELECT     dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_ID,
dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Test_RuleID,
                dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Test_ID,

dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Param1_ComputedType,

dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Param1_ComputedValue,

dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Param2_ComputedType,

dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Param2_ComputedValue,

dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Test_Rule_Result,

dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Test_Rule_Error,

dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Test_Rule_LastUpdate,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_ID,
                dbo.vFaultDetection_Tests_Rules.FaultDetection_Test_ID,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Group,
                dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Order,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param1,
                dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param2,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_OperationType,
                dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Negate,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_RelationType,
```

```
            dbo.vFaultDetection_Tests_Rules.FaultDetection_OperationTypeName,
dbo.vFaultDetection_Tests_Rules.FaultDetection_RelationName,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param1_Type,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param1_Value,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param1_ID,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param1_Name,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param2_Type,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param2_Value,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param2_ID,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_Param2_Name,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Test_Order,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Test_Name,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Test_SuccessMessage,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Test_FailMessage,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Case_ID,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Case_ModuleTypeID,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Case_Order,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Case_Name,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Case_Description,
dbo.vFaultDetection_Tests_Rules.FaultDetection_Case_Flags,
            dbo.vFaultDetection_Tests_Rules.FaultDetection_Case_Deleted,
dbo.vFaultDetection_Tests_Rules.FaultDetection_RelationType
FROM        dbo.FaultDetection_Executions_Tests_Rules INNER JOIN
            dbo.vFaultDetection_Tests_Rules ON
            dbo.FaultDetection_Executions_Tests_Rules.FaultDetection_Execution_Test_RuleID =
dbo.vFaultDetection_Tests_Rules.FaultDetection_Rule_ID
```

3. Stored Procedures

3.1 stp_External_FaultDetection_Get

Gets a complete fault detection case collection, notification teams, tests and their rules to execute a Fault Detection on a specific Module (by its Module Type Id)

```
/*************************************************************************
*************************************************************************
**********************
*     Name        :     [dbo].[stp_External_FaultDetection_Get]
*     Created           :     01/01/2014
*     Author            :     itay bar
*
      ========================================================================
====================================
*     Over View
*
      ========================================================================
====================================
*     Gets a complete fault detection case collection, notification teams,
tests and their params to execute a
*     Fault Detection on a specific Module (by its Module Type Id)
*
      ========================================================================
====================================
*     -- EXEC [stp_External_FaultDetection_Get] 2
*************************************************************************
*************************************************************************
```

```
**************************************/

ALTER PROCEDURE [dbo].[stp_External_FaultDetection_Get]
            @ModuleTypeId INT

AS
-----------------------------------
-- DECLARE VARIABLES
-----------------------------------
      DECLARE @ErrorCode varchar(1)
      DECLARE @ErrorNumber int
-----------------------------------
-- SET VARIABLES
-----------------------------------
      SET @ErrorNumber = 0
      SET @ErrorCode = 'S'
-------------------------------------
-- Execution
-------------------------------------
      IF (@ModuleTypeId < 1 or NOT EXISTS(SELECT
FaultDetection_ModuleTypeID FROM FaultDetection_ModuleTypes where
FaultDetection_ModuleTypeID = @ModuleTypeId)) BEGIN
            set @ErrorNumber = 3009
            set @ErrorCode = 'H'
      END

-------------------------------------
-- Return Results
-------------------------------------
      DECLARE @caseList TABLE (Id INT)

      insert into @caseList select FaultDetection_Case_ID from
vFaultDetection_Cases
            where FaultDetection_ModuleTypeID = @ModuleTypeId and
FaultDetection_Case_Deleted = 0
      -- CASES
      select * from vFaultDetection_Cases where FaultDetection_Case_ID in
(select Id FROM @caseList)
      -- NOTIFICATION TEAMS
      SELECT * from vFaultDetection_Cases_Notifications vFDCN where
(vFDCN.FaultDetection_Case_ID in (select Id FROM @caseList) and
vFDCN.FaultDetection_Case_Notification_Deleted = 0)
      -- TESTS
      select * from vFaultDetection_Tests where FaultDetection_Case_ID in
(select Id FROM @caseList)
            ORDER BY FaultDetection_Case_Order, FaultDetection_Test_Order
      -- TESTS PARAMS
      select * from vFaultDetection_Tests_Rules where
FaultDetection_Case_ID in (select Id FROM @caseList)
            ORDER BY FaultDetection_Case_Order, FaultDetection_Test_Order,
FaultDetection_Rule_Group, FaultDetection_Rule_Order
```

3.2 Past Executions Stored Procedures

3.2.1 stp_External_FaultDetection_Execution_Get

Gets a past FaultDetection execution by Id. Includes Execution Details, Module Properties, and All Cases, Tests and their Rules.

```
/*************************************************************************
*************************************************************************
**********************
*     Name        :     [dbo].[stp_External_FaultDetection_Execution_Get]
*     Created           :     01/01/2014
*     Author            :     itay bar
*
      ========================================================================
====================================
*     Over View
*
      ========================================================================
====================================
*     Gets a past FaultDetection execution by Id
*     Includes Execution Details, Module Properties, and All Cases, Tests
and their Params.
*
      ========================================================================
====================================
*     -- EXEC [stp_External_FaultDetection_Execution_Get] 224,1
*************************************************************************
*************************************************************************
************************************/

ALTER PROCEDURE [dbo].[stp_External_FaultDetection_Execution_Get]
            @ExecutionID INT,
            @FailedOnly INT = 0

AS
-----------------------------------
-- DECLARE VARIABLES
-----------------------------------
      DECLARE @ErrorCode varchar(1)
      DECLARE @ErrorNumber int
      DECLARE @ReturnExecutionID int
-----------------------------------
-- SET VARIABLES
-----------------------------------
      SET @ErrorNumber = 0
      SET @ErrorCode = 'S'
      SET @ReturnExecutionID = @ExecutionID
-------------------------------------
-- Execution
-------------------------------------
      IF (@ExecutionID < 1 or NOT EXISTS(SELECT FaultDetection_Execution_ID
FROM FaultDetection_Executions where FaultDetection_Execution_ID =
@ExecutionID)) BEGIN
            set @ErrorNumber = 3009
            set @ErrorCode = 'H'
      END

-------------------------------------
-- Return Results
-------------------------------------
```

```
      SELECT FDE.*, @ErrorCode as ErrorCode, @ErrorNumber as ErrorNumber,
E.ErrorDescription from ErrorCodes E
      left outer join vFaultDetection_Executions FDE on
(FDE.FaultDetection_Execution_ID = @ReturnExecutionID)
      where E.ErrorNumber = @ErrorNumber

      if (@ErrorNumber = 0) BEGIN
            SELECT FDEMP.*, @ErrorCode as ErrorCode, @ErrorNumber as
ErrorNumber, E.ErrorDescription from ErrorCodes E
            left outer join vFaultDetection_Executions_ModuleProperties
FDEMP on (FDEMP.FaultDetection_Execution_ID = @ExecutionID AND
FDEMP.FaultDetection_Execution_Deleted = 0)
            where E.ErrorNumber = @ErrorNumber
            IF (@FailedOnly = 0) BEGIN


                  SELECT FDEC.*, @ErrorCode as ErrorCode, @ErrorNumber as
ErrorNumber, E.ErrorDescription from ErrorCodes E
                  left outer join vFaultDetection_Executions_Cases FDEC on
(FDEC.FaultDetection_Execution_ID = @ExecutionID AND
FDEC.FaultDetection_Execution_Deleted = 0)
                  where E.ErrorNumber = @ErrorNumber

                  SELECT vFDCN.*, @ErrorCode as ErrorCode, @ErrorNumber as
ErrorNumber, E.ErrorDescription from ErrorCodes
E,vFaultDetection_Cases_Notifications vFDCN
                  WHERE E.ErrorNumber = @ErrorNumber
                        AND (vFDCN.FaultDetection_Case_ID in (select
FaultDetection_Execution_Case_ID from vFaultDetection_Executions_Cases FDEC
where (FDEC.FaultDetection_Execution_ID = @ExecutionID AND
FDEC.FaultDetection_Execution_Deleted = 0)))
                        and vFDCN.FaultDetection_Case_Notification_Deleted
= 0

                  SELECT FDET.*, @ErrorCode as ErrorCode, @ErrorNumber as
ErrorNumber, E.ErrorDescription from ErrorCodes E
                  left outer join vFaultDetection_Executions_Tests FDET on
(FDET.FaultDetection_Execution_ID = @ExecutionID AND
FDET.FaultDetection_Execution_Deleted = 0)
                  where E.ErrorNumber = @ErrorNumber
                        ORDER BY FDET.FaultDetection_Execution_Case_ID,
FDET.FaultDetection_Execution_Test_ID

                  SELECT FDETP.*, @ErrorCode as ErrorCode, @ErrorNumber as
ErrorNumber, E.ErrorDescription from ErrorCodes E
                  left outer join vFaultDetection_Executions_Tests_Rules
FDETP on (FDETP.FaultDetection_Execution_ID = @ExecutionID)
                  where E.ErrorNumber = @ErrorNumber
                        ORDER BY FDETP.FaultDetection_Rule_Order,
FDETP.FaultDetection_Rule_Group
            END
            ELSE BEGIN
                  SELECT FDEC.*, @ErrorCode as ErrorCode, @ErrorNumber as
ErrorNumber, E.ErrorDescription from ErrorCodes E
                  left outer join vFaultDetection_Executions_Cases FDEC on
(FDEC.FaultDetection_Execution_ID = @ExecutionID AND
FDEC.FaultDetection_Execution_Case_Result = 1 AND
FDEC.FaultDetection_Execution_Deleted = 0)
                  where E.ErrorNumber = @ErrorNumber

                  SELECT vFDCN.*, @ErrorCode as ErrorCode, @ErrorNumber as
```

```
ErrorNumber, E.ErrorDescription from ErrorCodes
E,vFaultDetection_Cases_Notifications vFDCN
                  WHERE E.ErrorNumber = @ErrorNumber
                        AND (vFDCN.FaultDetection_Case_ID in (select
FaultDetection_Execution_Case_ID from vFaultDetection_Executions_Cases FDEC
where (FDEC.FaultDetection_Execution_ID = @ExecutionID AND
FDEC.FaultDetection_Execution_Case_Result = 1 and
FDEC.FaultDetection_Execution_Deleted = 0)))
                        and vFDCN.FaultDetection_Case_Notification_Deleted
= 0


                  SELECT FDET.*, @ErrorCode as ErrorCode, @ErrorNumber as
ErrorNumber, E.ErrorDescription from ErrorCodes E
                  left outer join vFaultDetection_Executions_Tests FDET on
(FDET.FaultDetection_Execution_ID = @ExecutionID AND
FDET.FaultDetection_Execution_Test_Result = 1 AND
FDET.FaultDetection_Execution_Deleted = 0)
                  where E.ErrorNumber = @ErrorNumber
                        ORDER BY FDET.FaultDetection_Execution_Case_ID,
FDET.FaultDetection_Execution_Test_ID

                  SELECT FDETP.*, @ErrorCode as ErrorCode, @ErrorNumber as
ErrorNumber, E.ErrorDescription from ErrorCodes E
                  left outer join vFaultDetection_Executions_Tests_Rules
FDETP on (FDETP.FaultDetection_Execution_ID = @ExecutionID and
FDETP.FaultDetection_Execution_Test_Rule_Result = 1)
                  where E.ErrorNumber = @ErrorNumber
                        ORDER BY FDETP.FaultDetection_Test_ID,
FDETP.FaultDetection_Rule_Order, FDETP.FaultDetection_Rule_Group
            END
      END
```

3.2.2 stp_External_FaultDetection_Execution_Save

Saves or Updates a past Fault Detection Execution.

```
/*************************************************************************
*************************************************************************
**********************
*     Name        :     [dbo].[stp_External_FaultDetection_Execution_Save]
*     Created           :     01/01/2014
*     Author            :     itay bar
*
      ========================================================================
====================================
*     Over View
*
      ========================================================================
====================================
*     Saves or Updates a past Fault Detection Execution.
*
      ========================================================================
====================================
*     -- EXEC [stp_External_FaultDetection_Execution_Save] 7,2,'Module Id
2','Module Name 2','2014-01-23 17:00',200,0,NULL,0
*     -- EXEC [stp_External_FaultDetection_Execution_Save] 0,2,'Module
Id','Module Name','2014-01-23 17:00',200,0,NULL,0
*************************************************************************
*************************************************************************
************************************/
```

```
ALTER PROCEDURE [dbo].[stp_External_FaultDetection_Execution_Save]
            @ExecutionID INT = 0,
            @ExecutionModuleTypeId int,
            @ExecutionModuleUniqueId nvarchar(50),
            @ExecutionModuleUniqueName nvarchar(250),
            @ExecutionTime datetime = null,
            @ExecutionDuration INT,
            @ExecutionResult INT = 0,
            @ExecutionError nvarchar(500) = null,
            @Deleted int = 0

AS
-----------------------------------
-- DECLARE VARIABLES
-----------------------------------
      DECLARE @ErrorCode varchar(1)
      DECLARE @ErrorNumber int
      DECLARE @ReturnExecutionID int
-----------------------------------
-- SET VARIABLES
-----------------------------------
      SET @ErrorNumber = 0
      SET @ErrorCode = 'S'
      SET @ReturnExecutionID = @ExecutionID
-------------------------------------
-- Execution
-------------------------------------
      IF (@ExecutionID > 0) BEGIN
            IF (EXISTS(select FaultDetection_Execution_ID from
FaultDetection_Executions where FaultDetection_Execution_ID =
@ExecutionID))
            BEGIN
                  -- UPDATE TABLE
                  BEGIN TRY
                        UPDATE FaultDetection_Executions
                        SET FaultDetection_Execution_Module_TypeID =
@ExecutionModuleTypeId,
                              FaultDetection_Execution_Module_UniqueId =
@ExecutionModuleUniqueId,
                              FaultDetection_Execution_Module_UniqueName =
@ExecutionModuleUniqueName,
                              FaultDetection_Execution_Time =
@ExecutionTime,
                              FaultDetection_Execution_Duration =
@ExecutionDuration,
                              FaultDetection_Execution_Result =
@ExecutionResult,
                              FaultDetection_Execution_Error =
@ExecutionError,
                              FaultDetection_Execution_LastUpdate =
getdate(),
                              FaultDetection_Execution_Deleted = @Deleted
                        WHERE
                              FaultDetection_Execution_ID = @ExecutionID

                        SET @ReturnExecutionID = @ExecutionID
                        SET @ErrorNumber = 0
                        SET @ErrorCode = 'S'
                  END TRY
                  BEGIN CATCH
```

```
                        SET @ErrorNumber = 20002 -- Failed updaing
Execution Record
                        SET @ErrorCode = 'H'
                  END CATCH
            END
            ELSE BEGIN
                  SET @ErrorNumber = 20001 -- Execution ID Does not exist
in Exectuions Table
                  SET @ErrorCode = 'H'
            END
      END
      ELSE BEGIN
            -- INSERT TO TABLE
            BEGIN TRY
                  INSERT INTO FaultDetection_Executions
                        (FaultDetection_Execution_Module_TypeID,
FaultDetection_Execution_Module_UniqueId,
FaultDetection_Execution_Module_UniqueName,
                        FaultDetection_Execution_Time,
FaultDetection_Execution_Duration, FaultDetection_Execution_Result,
FaultDetection_Execution_Error,
                        FaultDetection_Execution_LastUpdate,
FaultDetection_Execution_Deleted)
                  VALUES (@ExecutionModuleTypeId, @ExecutionModuleUniqueId,
@ExecutionModuleUniqueName, @ExecutionTime, @ExecutionDuration,
                              @ExecutionResult, @ExecutionError, getdate(),
@Deleted)

                  SELECT @ReturnExecutionID = SCOPE_IDENTITY()
                  SET @ErrorNumber = 0
                  SET @ErrorCode = 'S'
            END TRY
            BEGIN CATCH
                  SET @ErrorNumber = 20003 -- Failed creating a new
Execution Record
                  SET @ErrorCode = 'H'
            END CATCH
      END
-------------------------------------
-- Return Results
-------------------------------------
      SELECT FDE.*, @ErrorCode as ErrorCode, @ErrorNumber as ErrorNumber,
E.ErrorDescription from ErrorCodes E
      left outer join vFaultDetection_Executions FDE on
(FDE.FaultDetection_Execution_ID = @ReturnExecutionID)
      where E.ErrorNumber = @ErrorNumber
```

3.2.3 stp_External_FaultDetection_Execution_Case_Save

Saves a Fault Detection Case (and its Result) from a past execution.

```
/*************************************************************************
*************************************************************************
**********************
*     Name        :
      [dbo].[stp_External_FaultDetection_Execution_Case_Save]
*     Created           :     01/01/2014
*     Author            :     itay bar
*
      ========================================================================
====================================
*     Over View
*
      ========================================================================
====================================
*     Saves an Fault Detection Case (and its Result) from a past execution.
*
      ========================================================================
====================================
*     -- EXEC [stp_External_FaultDetection_Execution_Case_Save] 1,1,0,NULL
*     -- EXEC [stp_External_FaultDetection_Execution_Case_Save]
1,1,1,'ERROR'
*************************************************************************
*************************************************************************
************************************/

ALTER PROCEDURE [dbo].[stp_External_FaultDetection_Execution_Case_Save]
            @ExecutionID INT,
            @CaseID int,
            @CaseResult int,
            @CaseError nvarchar(500) = NULL

AS
-----------------------------------
-- DECLARE VARIABLES
-----------------------------------
      DECLARE @ErrorCode varchar(1)
      DECLARE @ErrorNumber int
      DECLARE @ReturnExecutionID int
      DECLARE @ReturnCaseID int
-----------------------------------
-- SET VARIABLES
-----------------------------------
      SET @ErrorNumber = 0
      SET @ErrorCode = 'S'
      SET @ReturnExecutionID = @ExecutionID
      SET @ReturnCaseID = @CaseID
---------------------------------------
-- Execution
---------------------------------------
      IF (@ExecutionID < 1 or NOT EXISTS(select FaultDetection_Execution_ID
from FaultDetection_Executions where FaultDetection_Execution_ID =
@ExecutionID))
      BEGIN
            SET @ErrorNumber = 20001 -- Execution ID Does not exist in
Exectuions Table
            SET @ErrorCode = 'H'
      END
```

```
	ELSE IF (@CaseID < 1 or NOT EXISTS(select FaultDetection_Case_ID from
FaultDetection_Cases where FaultDetection_Case_ID = @CaseID))
	BEGIN
		SET @ErrorNumber = 20004 -- Case ID Does not exist in Cases
Table
		SET @ErrorCode = 'H'
	END
	ELSE BEGIN
		IF (EXISTS(select FaultDetection_Execution_Case_ID from
FaultDetection_Executions_Cases where FaultDetection_Execution_ID =
@ExecutionID and FaultDetection_Execution_Case_ID = @CaseID)) BEGIN
			-- UPDATE TABLE
			BEGIN TRY
				UPDATE FaultDetection_Executions_Cases
				SET FaultDetection_Execution_Case_Result =
@CaseResult,
					FaultDetection_Execution_Case_Error =
@CaseError,
					FaultDetection_Execution_Case_LastUpdate =
getdate()
				WHERE
					FaultDetection_Execution_ID = @ExecutionID
					and
					FaultDetection_Execution_Case_ID = @CaseID

				SET @ErrorNumber = 0
				SET @ErrorCode = 'S'
			END TRY
			BEGIN CATCH
				SET @ErrorNumber = 20005 -- Failed updaing
Execution Case Record
				SET @ErrorCode = 'H'
			END CATCH
		END ELSE BEGIN
			-- INSERT NEW RECORD
			BEGIN TRY
				INSERT INTO FaultDetection_Executions_Cases
					(FaultDetection_Execution_ID,
FaultDetection_Execution_Case_ID, FaultDetection_Execution_Case_Result,
					FaultDetection_Execution_Case_Error,
FaultDetection_Execution_Case_LastUpdate)
				VALUES (@ExecutionID, @CaseID, @CaseResult,
@CaseError, getdate())

				SET @ErrorNumber = 0
				SET @ErrorCode = 'S'
			END TRY
			BEGIN CATCH
				SET @ErrorNumber = 20006 -- Failed creating a new
Execution Case Record
				SET @ErrorCode = 'H'
			END CATCH
		END
	END

-------------------------------------
-- Return Results
-------------------------------------
	SELECT FDEC.*, @ErrorCode as ErrorCode, @ErrorNumber as ErrorNumber,
E.ErrorDescription from ErrorCodes E
	left outer join vFaultDetection_Executions_Cases FDEC on
```

```
(FDEC.FaultDetection_Execution_ID = @ReturnExecutionID AND
FDEC.FaultDetection_Execution_Case_ID = @ReturnCaseID)
      where E.ErrorNumber = @ErrorNumber
```

3.2.4 stp_External_FaultDetection_Execution_ModuleProperty_Save

Save a module property from a past execution.

```
/*************************************************************************
*************************************************************************
**********************
*     Name        :
      [dbo].[stp_External_FaultDetection_Execution_ModuleProperty_Save]
*     Created           :     01/01/2014
*     Author            :     itay bar
*
      ========================================================================
====================================
*     Over View
*
      ========================================================================
====================================
*     Save a module property from a past execution.
*
      ========================================================================
====================================
*     -- EXEC [stp_External_FaultDetection_Execution_ModuleProperty_Save]
5,'IP','1.1.1.2'
*     -- EXEC [stp_External_FaultDetection_Execution_ModuleProperty_Save]
231,'IP','1.1.1.1'
*************************************************************************
*************************************************************************
*************************************/

ALTER PROCEDURE
[dbo].[stp_External_FaultDetection_Execution_ModuleProperty_Save]
            @ExecutionID INT,
            @PropertyName nvarchar(250),
            @PropertyValue nvarchar(250)

AS
-----------------------------------
-- DECLARE VARIABLES
-----------------------------------
      DECLARE @ErrorCode varchar(1)
      DECLARE @ErrorNumber int
-----------------------------------
-- SET VARIABLES
-----------------------------------
      SET @ErrorNumber = 0
      SET @ErrorCode = 'S'
-------------------------------------
-- Execution
-------------------------------------
      IF (@ExecutionID < 1 or NOT EXISTS(select FaultDetection_Execution_ID
from FaultDetection_Executions where FaultDetection_Execution_ID =
@ExecutionID))
      BEGIN
            SET @ErrorNumber = 20001 -- Execution ID Does not exist in
Exectuions Table
            SET @ErrorCode = 'H'
```

```
	END
	ELSE BEGIN
		IF (EXISTS(select FaultDetection_Module_Property_Name from
FaultDetection_Executions_ModuleProperties where
FaultDetection_Execution_ID = @ExecutionID and
FaultDetection_Module_Property_Name = @PropertyName))
		BEGIN
			-- UPDATE TABLE
			BEGIN TRY
				UPDATE FaultDetection_Executions_ModuleProperties
				SET FaultDetection_Module_Property_Value =
@PropertyValue,
					FaultDetection_Module_Property_LastUpdate =
getdate()
				WHERE
					FaultDetection_Execution_ID = @ExecutionID
					and
					FaultDetection_Module_Property_Name =
@PropertyName

				SET @ErrorNumber = 0
				SET @ErrorCode = 'S'
			END TRY
			BEGIN CATCH
				SET @ErrorNumber = 20007 -- Failed updaing
Execution Case Record
				SET @ErrorCode = 'H'
			END CATCH
		END
		ELSE BEGIN
			-- INSERT NEW RECORD
			BEGIN TRY
				INSERT INTO
FaultDetection_Executions_ModuleProperties
					(FaultDetection_Execution_ID,
FaultDetection_Module_Property_Name, FaultDetection_Module_Property_Value,
FaultDetection_Module_Property_LastUpdate)
				VALUES (@ExecutionID, @PropertyName,
@PropertyValue, getdate())

				SET @ErrorNumber = 0
				SET @ErrorCode = 'S'
			END TRY
			BEGIN CATCH
				SET @ErrorNumber = 20008 -- Failed creating a new
Execution Case Record
				SET @ErrorCode = 'H'
			END CATCH
		END
	END

-------------------------------------
-- Return Results
-------------------------------------
	SELECT FDEMP.*, @ErrorCode as ErrorCode, @ErrorNumber as ErrorNumber,
E.ErrorDescription from ErrorCodes E
	left outer join vFaultDetection_Executions_ModuleProperties FDEMP on
(FDEMP.FaultDetection_Execution_ID = @ExecutionID AND
FDEMP.FaultDetection_Module_Property_Name = @PropertyName)
	where E.ErrorNumber = @ErrorNumber
```

3.2.5 stp_External_FaultDetection_Execution_Test_Save

Saves a test from a past Fault Detection execution.

```
/*************************************************************************
*************************************************************************
**********************
*     Name        :
      [dbo].[stp_External_FaultDetection_Execution_Test_Save]
*     Created           :     01/01/2014
*     Author            :     itay bar
*
      ========================================================================
====================================
*     Over View
*
      ========================================================================
====================================
*     Saves a test from a past Fault Detection execution.
*
      ========================================================================
====================================
*     -- EXEC [stp_External_FaultDetection_Execution_Test_Save]
5,'IP','1.1.1.2'
*     -- EXEC [stp_External_FaultDetection_Execution_Test_Save]
1,'IP','1.1.1.1'
*************************************************************************
*************************************************************************
************************************/

ALTER PROCEDURE [dbo].[stp_External_FaultDetection_Execution_Test_Save]
            @ExecutionID INT,
            @CaseID INT,
            @TestID INT,
            @TestOrder INT,
            @TestResult INT,
            @TestError nvarchar(500) = null

AS
-----------------------------------
-- DECLARE VARIABLES
-----------------------------------
      DECLARE @ErrorCode varchar(1)
      DECLARE @ErrorNumber int
-----------------------------------
-- SET VARIABLES
-----------------------------------
      SET @ErrorNumber = 0
      SET @ErrorCode = 'S'
--------------------------------------
-- Execution
--------------------------------------
      IF (@ExecutionID < 1 or NOT EXISTS(select FaultDetection_Execution_ID
from FaultDetection_Executions where FaultDetection_Execution_ID =
@ExecutionID))
      BEGIN
            SET @ErrorNumber = 20001 -- Execution ID Does not exist in
Exectuions Table
            SET @ErrorCode = 'H'
      END
      ELSE IF (@CaseID < 1 or NOT EXISTS(select FaultDetection_Case_ID from
```

```
FaultDetection_Cases where FaultDetection_Case_ID = @CaseID))
	BEGIN
		SET @ErrorNumber = 20004 -- Case ID Does not exist in Cases
Table
		SET @ErrorCode = 'H'
	END
	ELSE IF (@TestID < 1 or NOT EXISTS(select FaultDetection_Test_ID from
FaultDetection_Tests where FaultDetection_Test_ID = @TestID))
	BEGIN
		SET @ErrorNumber = 20009 -- Test ID does not exist in Tests
Table
		SET @ErrorCode = 'H'
	END
	ELSE BEGIN
		IF (EXISTS(select FaultDetection_Execution_Test_ID from
FaultDetection_Executions_Tests where FaultDetection_Execution_ID =
@ExecutionID and FaultDetection_Execution_Case_ID = @CaseID and
FaultDetection_Execution_Test_ID = @TestID))
		BEGIN
			-- UPDATE TABLE
			BEGIN TRY
				UPDATE FaultDetection_Executions_Tests
				SET FaultDetection_Execution_Test_Order =
@TestOrder,
					FaultDetection_Execution_Test_Result =
@TestResult,
					FaultDetection_Execution_Test_Error =
@TestError,
					FaultDetection_Execution_Test_LastUpdate =
getdate()
				WHERE
					FaultDetection_Execution_ID = @ExecutionID
					and
					FaultDetection_Execution_Case_ID = @CaseID
					and
					FaultDetection_Execution_Test_ID = @TestID

				SET @ErrorNumber = 0
				SET @ErrorCode = 'S'
			END TRY
			BEGIN CATCH
				SET @ErrorNumber = 20010 -- Failed updaing
Execution Test Record
				SET @ErrorCode = 'H'
			END CATCH
		END
		ELSE BEGIN
			-- INSERT NEW RECORD
			BEGIN TRY
				INSERT INTO FaultDetection_Executions_Tests
					(FaultDetection_Execution_ID,
FaultDetection_Execution_Case_ID, FaultDetection_Execution_Test_Order,
FaultDetection_Execution_Test_ID, FaultDetection_Execution_Test_Result,
					FaultDetection_Execution_Test_Error,
FaultDetection_Execution_Test_LastUpdate)
				VALUES (@ExecutionID, @CaseID, @TestOrder, @TestID,
@TestResult, @TestError, getdate())

				SET @ErrorNumber = 0
				SET @ErrorCode = 'S'
			END TRY
```

```
                  BEGIN CATCH
                        SET @ErrorNumber = 20011 -- Failed creating a new
Execution Test Record
                        SET @ErrorCode = 'H'
                  END CATCH
            END
      END

-------------------------------------
-- Return Results
-------------------------------------
      SELECT FDET.*, @ErrorCode as ErrorCode, @ErrorNumber as ErrorNumber,
E.ErrorDescription from ErrorCodes E
      left outer join vFaultDetection_Executions_Tests FDET on
(FDET.FaultDetection_Execution_ID = @ExecutionID AND
FDET.FaultDetection_Execution_Case_ID = @CaseID AND
FDET.FaultDetection_Execution_Test_ID = @TestID)
      where E.ErrorNumber = @ErrorNumber
```

3.2.6 stp_External_FaultDetection_Execution_Test_Rule_Save

Saves a Test Rule from a Past Execution (with its results and parameters computed values from the time of that execution).

```
/*************************************************************************
*************************************************************************
**********************
*     Name        :
      [dbo].[stp_External_FaultDetection_Execution_Test_Rule_Save]
*     Created           :     01/01/2014
*     Author            :     itay bar
*
      ========================================================================
====================================
*     Over View
*
      ========================================================================
====================================
*     Saves a Test Rule from a Past Execution (with its results and
parameters computed values from the time of that execution).
*
      ========================================================================
====================================
*     -- EXEC [stp_External_FaultDetection_Execution_Test_Rule_Save]
5,'IP','1.1.1.2'
*     -- EXEC [stp_External_FaultDetection_Execution_Test_Rule_Save]
231,1,1,1,'value1',1,'value2',0,NULL
*************************************************************************
*************************************************************************
************************************/

ALTER PROCEDURE
[dbo].[stp_External_FaultDetection_Execution_Test_Rule_Save]
            @ExecutionID INT,
            @RuleID INT,
            @TestID INT,
            @Param1ComputedType INT,
            @Param1ComputedValue nvarchar(250),
            @Param2ComputedType INT,
            @Param2ComputedValue nvarchar(250),
            @RuleResult INT,
```

```
            @RuleError nvarchar(500) = NULL

AS
-----------------------------------
-- DECLARE VARIABLES
-----------------------------------
      DECLARE @ErrorCode varchar(1)
      DECLARE @ErrorNumber int
-----------------------------------
-- SET VARIABLES
-----------------------------------
      SET @ErrorNumber = 0
      SET @ErrorCode = 'S'
-------------------------------------
-- Execution
-------------------------------------
      IF (@ExecutionID < 1 or NOT EXISTS(select FaultDetection_Execution_ID
from FaultDetection_Executions where FaultDetection_Execution_ID =
@ExecutionID))
      BEGIN
            SET @ErrorNumber = 20001 -- Execution ID Does not exist in
Exectuions Table
            SET @ErrorCode = 'H'
      END
      ELSE IF (@RuleID < 1 or NOT EXISTS(select FaultDetection_Rule_ID from
FaultDetection_Tests_Rules where FaultDetection_Rule_ID = @RuleID))
      BEGIN
            SET @ErrorNumber = 20012 -- Rule ID does not exist in Tests
Rules Table
            SET @ErrorCode = 'H'
      END
      ELSE IF (@TestID < 1 or NOT EXISTS(select FaultDetection_Test_ID from
FaultDetection_Tests where FaultDetection_Test_ID = @TestID))
      BEGIN
            SET @ErrorNumber = 20009 -- Test ID does not exist in Tests
Table
            SET @ErrorCode = 'H'
      END
      ELSE BEGIN
            IF (EXISTS(select FaultDetection_Execution_Test_RuleID from
FaultDetection_Executions_Tests_Rules where FaultDetection_Execution_ID =
@ExecutionID and FaultDetection_Execution_Test_RuleID = @RuleID and
FaultDetection_Execution_Test_ID = @TestID))
            BEGIN
                  -- UPDATE TABLE
                  BEGIN TRY
                        UPDATE FaultDetection_Executions_Tests_Rules
                        SET FaultDetection_Execution_Param1_ComputedType =
@Param1ComputedType,
                              FaultDetection_Execution_Param1_ComputedValue
= @Param1ComputedValue,
                              FaultDetection_Execution_Param2_ComputedType
= @Param2ComputedType,
                              FaultDetection_Execution_Param2_ComputedValue
= @Param2ComputedValue,
                              FaultDetection_Execution_Test_Rule_Result =
@RuleResult,
                              FaultDetection_Execution_Test_Rule_Error =
@RuleError,
                              FaultDetection_Execution_Test_Rule_LastUpdate
= getdate()
```

```
                        WHERE
                              FaultDetection_Execution_ID = @ExecutionID
                              and
                              FaultDetection_Execution_Test_RuleID =
@RuleID
                              and
                              FaultDetection_Execution_Test_ID = @TestID

                        SET @ErrorNumber = 0
                        SET @ErrorCode = 'S'
                  END TRY
                  BEGIN CATCH
                        SET @ErrorNumber = 20013 -- Failed updaing
Execution Test Params Record
                        SET @ErrorCode = 'H'
                  END CATCH
            END
            ELSE BEGIN
                  -- INSERT NEW RECORD
                  BEGIN TRY
                        INSERT INTO FaultDetection_Executions_Tests_Rules
                              (FaultDetection_Execution_ID,
FaultDetection_Execution_Test_RuleID, FaultDetection_Execution_Test_ID,
FaultDetection_Execution_Param1_ComputedType,

      FaultDetection_Execution_Param1_ComputedValue,
FaultDetection_Execution_Param2_ComputedType,FaultDetection_Execution_Param
2_ComputedValue,
                              FaultDetection_Execution_Test_Rule_Result,
FaultDetection_Execution_Test_Rule_Error,
FaultDetection_Execution_Test_Rule_LastUpdate)
                        VALUES (@ExecutionID, @RuleID, @TestID,
@Param1ComputedType, @Param1ComputedValue, @Param2ComputedType,
@Param2ComputedValue,
                                    @RuleResult, @RuleError, getdate())

                        SET @ErrorNumber = 0
                        SET @ErrorCode = 'S'
                  END TRY
                  BEGIN CATCH
                        SET @ErrorNumber = 20014 -- Failed creating
Execution Test Params Record
                        SET @ErrorCode = 'H'
                  END CATCH
            END
      END

-------------------------------------
-- Return Results
-------------------------------------
      SELECT FDETP.*, @ErrorCode as ErrorCode, @ErrorNumber as ErrorNumber,
E.ErrorDescription from ErrorCodes E
      left outer join vFaultDetection_Executions_Tests_Rules FDETP on
(FDETP.FaultDetection_Execution_ID = @ExecutionID AND
FDETP.FaultDetection_Execution_Test_RuleID = @RuleID AND
FDETP.FaultDetection_Execution_Test_ID = @TestID)
      where E.ErrorNumber = @ErrorNumber
```

Appendix B – Code Listing

The below code listing only include code that was written for this solution and disregards any automatically generated code by Visual Studio.

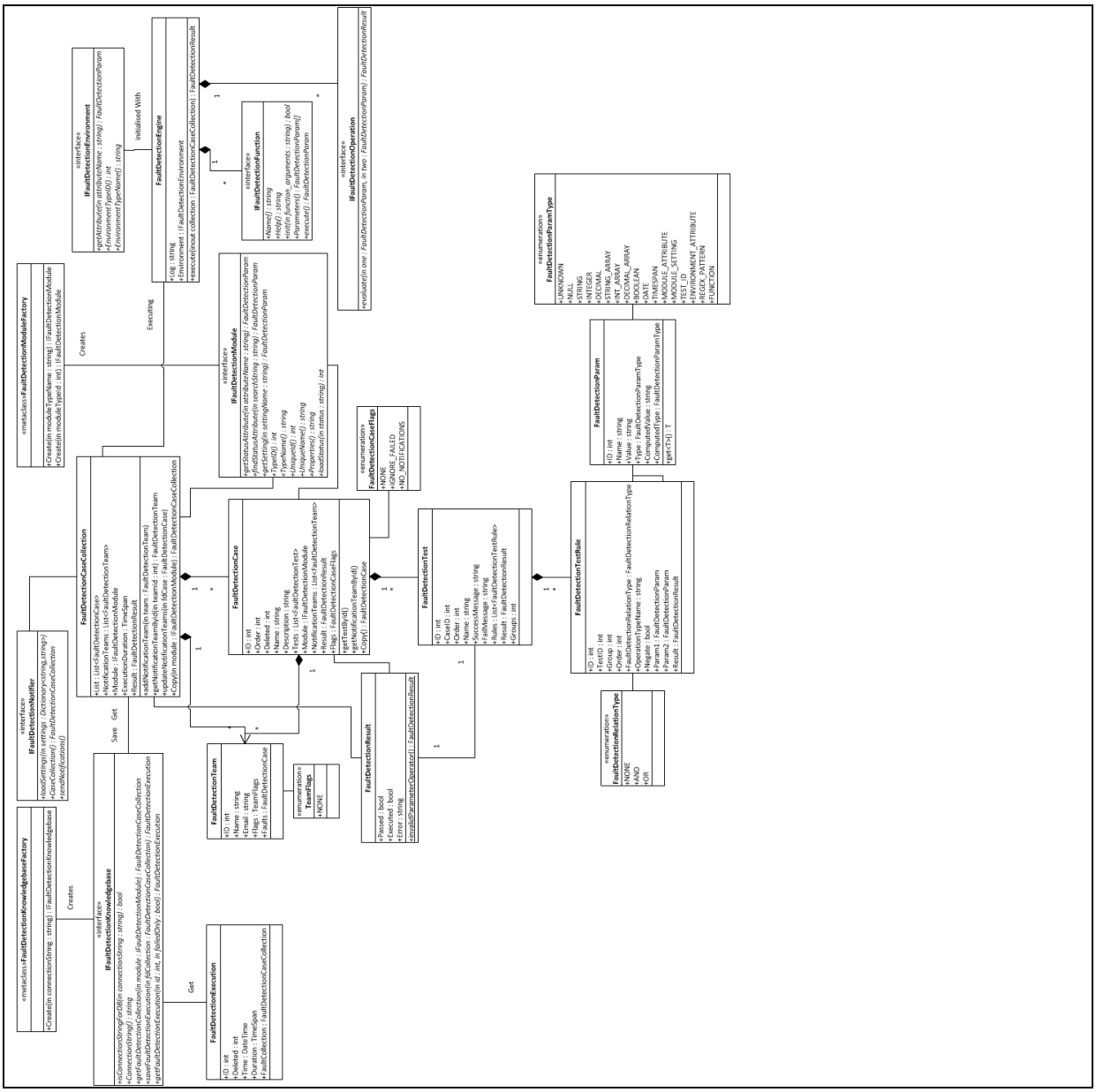
All class diagrams given below were generated using Microsoft Visual Studio and Microsoft Visio. Only public properties or methods are included. Note that C# interfaces allow for properties, these however were represented as methods in the Visio Class Diagram (1.1) since UML interface diagrams only support methods.

All the code listed below is included in Appendix D – Included Disc.

1. Fault Detection and Isolation (FDI) System

1.1 Class Diagram

The original Microsoft Visio (VSD) file of the below diagram is included in Appendix D – Included Disc.



1.2 Implementation/Concrete Classes

1.2.1 FaultDetectionCase.cs

Represents a "possible" Fault and has a List of Tests to execute to detect if this possible fault is present or not. If after execution, Result is not "Passed" this possible fault has been detected.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a "possible" Fault and has a List of Tests to execute to detect if
this possible fault is present or not.
    /// If after execution, Result is not "Passed" this possible fault has been
detected.
    /// </summary>
    public class FaultDetectionCase
    {

        private int _id, _order, _deleted;
        private String _name, _description;
        private List<FaultDetectionTest> _tests = new List<FaultDetectionTest>(1);
        private IFaultDetectionModule _module;
        private List<FaultDetectionTeam> _notificationTeams = new
List<FaultDetectionTeam>(1);
        private FaultDetectionResult _result = new FaultDetectionResult();
```

```
        private FaultDetectionCaseFlags _flags = FaultDetectionCaseFlags.NONE;

        public int ID {
            get { return this._id; }
            set { this._id = value; }
        }
        public int Order {
            get { return this._order; }
            set { this._order = value; }
        }
        public int Deleted
        {
            get { return this._deleted; }
            set { this._deleted = value; }
        }

        public String Name {
            get { return this._name; }
            set { this._name = value; }
        }
        public String Description
        {
            get { return this._description; }
            set { this._description = value; }
        }

        public List<FaultDetectionTest> Tests
        {
            get { return this._tests; }
            set { this._tests = value; }
        }

        public IFaultDetectionModule Module
        {
            get { return this._module; }
            set { this._module = value; }
        }

        public List<FaultDetectionTeam> NotificationTeams
        {
            get { return this._notificationTeams; }
            set { this._notificationTeams = value; }
        }

        public FaultDetectionResult Result
        {
            get { return this._result; }
            set { this._result = value; }
        }

        public FaultDetectionCaseFlags Flags {
            get { return this._flags; }
            set { this._flags = value; }
        }

        [Flags]
        public enum FaultDetectionCaseFlags : int
        {
            NONE = 0,
            // if set, the case will always be set as "passed" - this is used for
cases that have "supporting" or precondition type tests were failures should be
ignored.
```

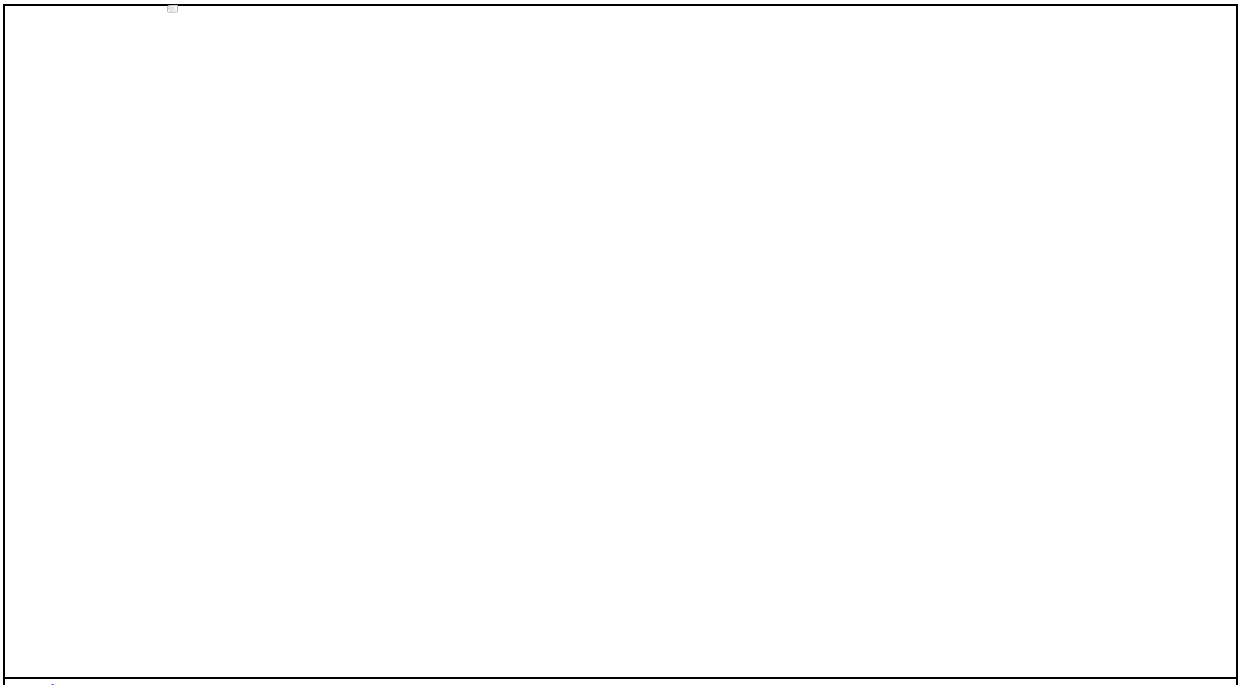
```
            IGNORE_FAILED = 1,
            // if SET, the case will not cause or be added to notifications - this is
also used for cases that have "supporting" or precondition type tests.
            NO_NOTIFICATIONS = 2
        }

        /// <summary>
        /// All classes must have a parameterless constructor so they can be
serializable.
        /// <para>
        /// Should not be used manually.
        /// </para>
        /// </summary>
        public FaultDetectionCase()
        {
        }

        /// <summary>
        /// Creates a new Fault Detection Case Object for a given Module.
        /// </summary>
        /// <param name="module">The Module the case is for</param>
        public FaultDetectionCase(IFaultDetectionModule module)
        {
            this.Module = module;
        }

        /// <summary>
        /// Creates a new Fault Detection Case Object for a given Module.
        /// </summary>
        /// <param name="module">The Module the case is for</param>
        /// <param name="id">The Case Unique ID</param>
        /// <param name="order">The cases Order (if in a collection)</param>
        /// <param name="name">The Case Name</param>
        /// <param name="description">The Case Description</param>
        /// <param name="flags">The Case Flags (FaultDetectionCaseFlags ENUM)</param>
        public FaultDetectionCase(IFaultDetectionModule module, int id, int order,
string name, string description, FaultDetectionCaseFlags flags) : this(module)
        {

            this.ID = id;
            this.Order = order;
            this.Name = name;
            this.Description = description;
            this.Flags = flags;
        }

        /// <summary>
        /// Gets a Fault Detection Test assosicated with this Fault Detection Case by
the Tests' Id.
        /// </summary>
        /// <param name="testId">the test id to get</param>
        /// <returns>FaultDetectionTest object if found, null otherwise</returns>
        public FaultDetectionTest getTestById(int testId)
        {
            foreach (FaultDetectionTest test in Tests)
            {
                if (test.ID == testId)
                {
                    return test;
                }
            }
            return null;
```

```
        }

        /// <summary>
        /// Gets a FaultDetectionTeam assosicted with this Fault Detection Case by the
Teams' Id.
        /// </summary>
        /// <param name="teamId">the team id of the notification team to get</param>
        /// <returns>FaultDetectionTeam object if found, null otherwise</returns>
        public FaultDetectionTeam getNotificationTeamById(int teamId)
        {
            foreach (FaultDetectionTeam team in NotificationTeams)
            {
                if (team.ID == teamId)
                {
                    return team;
                }
            }
            return null;
        }

        /// <summary>
        /// Copies the case without its results to a new module.
        /// <para>
        /// The given module must of the same type of module of this case.
        /// </para>
        /// </summary>
        /// <param name="module">the new module</param>
        /// <param name="includeTests">whether or not to copy this fault case
tests</param>
        /// <param name="setParamsComputed">whether or not to set the tests rule
parameterss computed value</param>
        /// <param name="includeTeams">whether ot not o copy the notification
teams</param>
        /// <returns>a new FaultDetectionCase for the given module</returns>
        /// <exception cref="ArgumentException">Thrown when the given Module is not
the same type of Module of this case</exception>
        public FaultDetectionCase Copy(IFaultDetectionModule module, bool
includeTests, bool setParamsComputed, bool includeTeams)
        {
            if (this.Module.TypeID != module.TypeID)
            {
                throw new ArgumentException("Module " + module.TypeID + " / " +
module.TypeName + " does not match case module (" + this.Module.TypeID + " / " +
this.Module.TypeName + ")");
            }
            FaultDetectionCase nfdCase = new FaultDetectionCase(module, this.ID,
this.Order, this.Name, this.Description, this.Flags);
            if (includeTests)
            {
                foreach (FaultDetectionTest test in this.Tests)
                {
                    nfdCase.Tests.Add(test.Copy(true,setParamsComputed));
                }
            }
            if (includeTeams) {
                foreach (FaultDetectionTeam cteam in this.NotificationTeams)
                {
                    nfdCase.NotificationTeams.Add(new FaultDetectionTeam(cteam.ID,
cteam.Name, cteam.Email, cteam.Flags));
                }
            }
            return nfdCase;
```

```
        }

    }
}
```

1.2.2 FaultDetectionCaseCollection.cs

Represents a collection of possible faults for a specific module. If after execution Result is not "passed", one of the possible faults in the collection has been detected.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Newtonsoft.Json;

namespace FaultDetection
{
    /// <summary>
    /// Represents a collection of possible faults for a specific module.
    /// If after execution Result is not "passed", one of the possible faults in the
collection has been detected.
    /// </summary>
    public class FaultDetectionCaseCollection
    {
        private List<FaultDetectionCase> _list = new List<FaultDetectionCase>(1);
        private List<FaultDetectionTeam> _notificationTeams = new
List<FaultDetectionTeam>(1);
        private IFaultDetectionModule _module;
        private TimeSpan _executionDuration = new TimeSpan(0);
        private FaultDetectionResult _result = new FaultDetectionResult();

        public List<FaultDetectionCase> List
        {
            get { return this._list; }
            set { this._list = value; }
        }

        public List<FaultDetectionTeam> NotificationTeams
```

```
        {
            get { return this._notificationTeams; }
            set { this._notificationTeams = value; }
        }

        public IFaultDetectionModule Module
        {
            get { return this._module; }
            set { this._module = value; }
        }

        public TimeSpan ExecutionDuration
        {
            get { return this._executionDuration; }
            set { this._executionDuration = value; }
        }

        public FaultDetectionResult Result
        {
            get { return this._result; }
            set { this._result = value; }
        }
        /// <summary>
        /// All classes must have a parameterless constructor so they can be
serializable.
        /// Should not be used manually.
        /// </summary>
        public FaultDetectionCaseCollection()
        {
        }
        /// <summary>
        /// Creates a new Fault Detection Case Collection for a given Module.
        /// </summary>
        /// <param name="module">The Module the case collection is for</param>
        public FaultDetectionCaseCollection(IFaultDetectionModule module)
        {
            this.Module = module;
        }

        /// <summary>
        /// Adds (if not already exists) a FaultDetectionTeam for notifications.
        /// </summary>
        /// <param name="team">the FaultDetectionCaseTeam to add</param>
        public void addNotificationTeam(FaultDetectionTeam team)
        {
            if (team != null && team.ID > 0)
            {
                foreach (FaultDetectionTeam nteam in this.NotificationTeams)
                {
                    if (nteam.ID == team.ID)
                    {
                        return;
                    }
                }
                this.NotificationTeams.Add(team);
            }
        }
        /// <summary>
        /// Gets a FaultDetectionTeam from the Notification Teams registered for this
Collection (i.e. registered to at least one of the cases in this collection).
        /// </summary>
        /// <param name="teamId">The team id of the FaultDetectionTeam to get</param>
```

```
        /// <returns>FaultDetectionTeam or null if not found</returns>
        public FaultDetectionTeam getNotificationTeamById(int teamId)
        {
            foreach (FaultDetectionTeam team in NotificationTeams)
            {
                if (team.ID == teamId)
                {
                    return team;
                }
            }
            return null;
        }
        /// <summary>
        /// Updates all teams registered for the given FaultDetectionCase when a fault
has been detected.
        /// <para>
        /// This FaultDetectionCase will the be sent (via email) as part of the alert.
        /// </para>
        /// </summary>
        /// <param name="fdCase">A FaultDetectionCase (which failed and detected as a
fault) to add</param>
        public void updateNotificationTeams(FaultDetectionCase fdCase)
        {
            if (fdCase != null && ((fdCase.Flags &
FaultDetectionCase.FaultDetectionCaseFlags.NO_NOTIFICATIONS) == 0) && fdCase.Result !=
null && fdCase.Result.Executed && !fdCase.Result.Passed)
            {
                foreach (FaultDetectionTeam team in fdCase.NotificationTeams)
                {
                    FaultDetectionTeam exteam = getNotificationTeamById(team.ID);
                    if (exteam != null)
                    {
                        exteam.Faults.Add(fdCase);
                    }
                }
            }
        }
        /// <summary>
        /// Copies the entire collection without its results to a new module.
        /// <para>
        /// The given module must of the same type of module of this case.
        /// </para>
        /// </summary>
        /// <param name="module">the new module</param>
        /// <returns>a new FaultDetectionCaseCollection for the given module</returns>
        /// /// <exception cref="ArgumentException">Thrown when the given Module is
not the same type of Module of this collection</exception>
        public FaultDetectionCaseCollection Copy(IFaultDetectionModule module)
        {
            if (this.Module.TypeID != module.TypeID)
            {
                throw new ArgumentException("Module " + module.TypeID + " / " +
module.TypeName + " does not match collection module (" + this.Module.TypeID + " / " +
this.Module.TypeName + ")");
            }
            FaultDetectionCaseCollection ncollection = new
FaultDetectionCaseCollection(module);
            foreach (FaultDetectionTeam team in this.NotificationTeams)
            {
                ncollection.NotificationTeams.Add(new FaultDetectionTeam(team.ID,
team.Name, team.Email, team.Flags));
            }
```

```
            foreach (FaultDetectionCase fdCase in this.List)
            {
                ncollection.List.Add(fdCase.Copy(module, true, false, true));
            }
            return ncollection;
        }
    }
}
```

1.2.3 FaultDetectionCaseTeam.cs

Extended the FaultDetectionTeam Class to include a CaseId. Used to quickly attach teams to a FaultDetectionCase when loading.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Extended the FaultDetectionTeam Class to include a CaseId.
    /// Used to quickly attach teams to a FaultDetectionCase when loading.
    /// </summary>
    public class FaultDetectionCaseTeam : FaultDetectionTeam
    {
        private int _caseId;

        public int CaseID
        {
            get { return this._caseId; }
            set { this._caseId = value; }
        }
    }
}
```

1.2.4 FaultDetectionKnowledgebaseFactory.cs

Used to create new instances of IFaultDetectionKnowledgebase based on a Connection String.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using FaultDetection.Knowledgebases.Sql;
using System.Reflection;
using System.Text.RegularExpressions;


namespace FaultDetection
{
    /// <summary>
    /// Used to create new instances of IFaultDetectionKnowledgebase based on a
Connection String.
    /// </summary>
    public static class FaultDetectionKnowledgebaseFactory
    {
        /// <summary>
        /// Creates a new instance of IFaultDetectionKnowledgebase based on a given
Connection String.
        /// <para>
        /// If no matching implementation found in Knowledgebases namespace for the
given ConnectionString, will return null.
        /// </para>
        /// </summary>
        /// <param name="connectionString">the connection string used to connect to
the knowledgebase (database)</param>
        /// <returns>IFaultDetectionKnowledgebase instance or null</returns>
        public static IFaultDetectionKnowledgebase Create(string connectionString)
        {
            Assembly pAssembly = Assembly.GetAssembly(typeof(FaultDetectionEngine));
            if (pAssembly != null)
            {
                // searching for all the types in "FaultDetection.Knowledgebases"
namespace (and sub-namespaces)
                foreach (Type pType in pAssembly.GetTypes().Where(t =>
Regex.IsMatch(t.FullName, @"FaultDetection\.Knowledgebases\..*")).ToArray())
                {
                    Type pInterface =
pType.GetInterface(typeof(IFaultDetectionKnowledgebase).ToString());
                    // if this type implements the IFaultDetectionKnowledgebase
interface, it will not be null
                    if (pInterface != null)
                    {
```

```
                        // creating a new IFaultDetectionKnowledgebase object
                        ConstructorInfo pConstructor = pType.GetConstructor(new Type[]
{ });
                        IFaultDetectionKnowledgebase pInstance =
pConstructor.Invoke(null) as IFaultDetectionKnowledgebase;
                        if (pInstance != null)
                        {
                            // check if the given connection string is for this
implementation of IFaultDetectionKnowledgebase, if so, sets the connection string and
returns the object.
                            if (pInstance.isConnectionStringForDB(connectionString))
                            {
                                pInstance.ConnectionString = connectionString;
                                return pInstance;
                            }
                        }
                    }
                }
            }
            return null;

        }
    }
}
```

1.2.5 FaultDetectionEngine.cs

The Fault Detection Engine used to execute a Fault Detection and Isolation Check.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using FaultDetection.Functions;
using System.Reflection;

namespace FaultDetection
{
    /// <summary>
    /// The Fault Detection Engine used to execute a Fault Detection and Isolation
Check.
    /// </summary>
    public class FaultDetectionEngine
    {
        private const string FUNCTION_PATTERN = @"(.*?)\((.*)\)";
        private List<IFaultDetectionFunction> _loadedFunctions = new
List<IFaultDetectionFunction>(1);
        private List<IFaultDetectionOperation> _loadedOperations = new
List<IFaultDetectionOperation>(1);
```

```
        private IFaultDetectionEnvironment _environment;

        private StringBuilder _log = new StringBuilder();

        public IFaultDetectionEnvironment Environment
        {
            get { return this._environment; }
            set { this._environment = value; }
        }

        public string Log
        {
            get { return this._log.ToString(); }
        }

        /// <summary>
        /// Initialises a FaultDetectionEngine object and loads all Functions and
Operations available
        /// </summary>
        public FaultDetectionEngine()
        {
            loadFunctions();
            loadOperations();
        }

        /// <summary>
        /// Initialises a FaultDetectionEngine object and loads all Functions and
Operations available
        /// <para>
        /// Sets the execution environment to the given IFaultDetectionEnviornment
        /// </para>
        /// </summary>
        /// <param name="environment">an IFaultDetectionEnvironment object
representing the environment the execution is preformed at</param>
        public FaultDetectionEngine(IFaultDetectionEnvironment environment) : this()
        {
            this.Environment = environment;
        }

        /// <summary>
        /// Executes a Fault Detection check based on a FaultDetectionCaseCollection
given.
        /// <para>
        /// The collection given will be changed during the execution, and all the
cases, tests and their parameter results will be set according to the execution
result.
        /// </para>
        /// </summary>
        /// <param name="collection">the collection of Fault Detection Cases to
execute</param>
        /// <returns>FaultDetectionResult representing the result of all the Fault
Detection Cases checked</returns>
        public FaultDetectionResult execute(ref FaultDetectionCaseCollection
collection)
        {
            _log.Clear();
            long ticksAtStart = DateTime.Now.Ticks;
            FaultDetectionResult tstr = new FaultDetectionResult();
            bool all = true; // represents whether or not all cases in this collection
passed
            if (collection != null)
```

```
            {
                for (int i = 0; i < collection.List.Count; i++)
                {
                    bool tsall = true; // represents whether or not all tests in this
case passed
                    for (int j = 0; j < collection.List[i].Tests.Count; j++)
                    {
                        FaultDetectionTest test = collection.List[i].Tests[j];
                        tstr = executeTest(ref collection, ref test, true);
                        if (!tstr.Passed)
                        {
                            all = false;
                            tsall = false;
                        }
                    }
                    collection.List[i].Result.Passed = ((collection.List[i].Flags &
FaultDetectionCase.FaultDetectionCaseFlags.IGNORE_FAILED) > 0 ? true : tsall); // if
case is an "ignore failed" case, set to true, otherwise set to the result of the
execution.
                    collection.updateNotificationTeams(collection.List[i]); // update
notification teams, if the case failed it will be added to the registered teams
CaseCollection and will be added to the notification (email) sent.
                }
            }
            if (all)
            {
                tstr.Passed = true;
            }
            else
            {
                tstr.Passed = false;
                tstr.Error = "At least 1 check failed";
            }
            collection.Result.Passed = tstr.Passed;
            if (!collection.Result.Passed)
            {
                collection.Result.Error = tstr.Error;
            }
            collection.ExecutionDuration = new TimeSpan(DateTime.Now.Ticks -
ticksAtStart);
            return tstr;
        }

        /// <summary>
        /// Executes a single test.
        /// <para>
        /// The collection object is used incase the test directs or uses a different
test from the collection.
        /// </para>
        /// </summary>
        /// <param name="collection">FaultDetectionCaseCollection used for
refrenceing/getting other tests results</param>
        /// <param name="test">the FaultDetectionTest to execute</param>
        /// <returns>FaultDetectionResult of the test execution</returns>
        private FaultDetectionResult executeTest(ref FaultDetectionCaseCollection
collection, ref FaultDetectionTest test)
        {
            return executeTest(ref collection, ref test, false);
        }

        /// <summary>
        /// Executes a single test.
```

```
        /// <para>
        /// The collection object is used incase the test directs or uses a different
test from the collection.
        /// </para>
        /// </summary>
        /// <param name="collection">FaultDetectionCaseCollection used for
refrenceing/getting other tests results</param>
        /// <param name="test">the FaultDetectionTest to execute</param>
        /// <param name="skipIfTested">Whether or not to skip a test it was already
executed (i.e. test.Result.Executed == true)</param>
        /// <returns>FaultDetectionResult of the test execution</returns>
        private FaultDetectionResult executeTest(ref FaultDetectionCaseCollection
collection, ref FaultDetectionTest test, bool skipIfTested)
        {
            if (skipIfTested && test.Result.Executed)
            {
                return test.Result;
            }
            FaultDetectionResult res = new FaultDetectionResult(false);
            bool all = false; // whether or not all sub-tests/groups passed
            bool lastMerged = true; // whether or not the last group was merged
            bool[] groups = new bool[test.Groups]; // each group passed state
            int lastGroup = 0;
            FaultDetectionRelationType lastGroupRelation = 0, lastRelation = 0;
            StringBuilder testLog = new StringBuilder(); // the test log, appended to
the execution log at the end of the test execution.
            testLog.AppendLine(test.ID + " / " + test.Name + ": \r\n(");
            if (test.Rules != null && test.Rules.Count > 0)
            {
                test.Rules[0].Result = evaluate(ref collection, test.Rules[0].Param1,
test.Rules[0].Param2, test.Rules[0].OperationTypeName);
                groups[test.Rules[0].Group] = test.Rules[0].Result.Passed;
                all = groups[test.Rules[0].Group];
                testLog.AppendLine("[i=0/g=" + test.Rules[0].Group + "/all=" + all +
"] { (" + getParamDetails(test.Rules[0].Param1) + " " + (test.Rules[0].Negate ? "NOT "
: "") + test.Rules[0].OperationTypeName + " " + getParamDetails(test.Rules[0].Param2)
+ ")");
                lastRelation = test.Rules[0].RelationType;
                lastGroupRelation = test.Rules[0].RelationType;
                for (int i = 1; i < test.Rules.Count; i++)
                {
                    FaultDetectionTestRule rule = test.Rules[i];
                    if (rule.Group != lastGroup)
                    {
                       testLog.AppendLine(" (group result: " + groups[lastGroup] + ")
} \r\n " + test.Rules[i - 1].RelationType + "\r\n (");
                    }
                    rule.Result = evaluate(ref collection, rule.Param1, rule.Param2,
rule.OperationTypeName);
                    testLog.AppendLine("[i=" + i + "/g=" + rule.Group + "/lg=" +
lastGroup + "/lg_res=" + groups[rule.Group] + "/all=" + all + "/last_relation=" +
lastRelation + "] " + ((rule.Group == lastGroup) ? lastRelation.ToString() : "") + "
(" + getParamDetails(rule.Param1) + " " + (rule.Negate ? "NOT " : "") +
rule.OperationTypeName + " " + getParamDetails(rule.Param2) + ") -> RESULT: " +
rule.Result.Passed);
                    lastMerged = false;
                    if (rule.Group == lastGroup)
                    {
                        groups[rule.Group] = merge(groups[rule.Group],
rule.Result.Passed, lastRelation);
                    }
                    else
```

```
                    {
                        lastGroupRelation = test.Rules[i - 1].RelationType;
                        all = merge(all, groups[lastGroup], lastGroupRelation);
                        groups[rule.Group] = rule.Result.Passed;
                        lastMerged = true;
                    }
                    lastRelation = rule.RelationType;
                    lastGroup = rule.Group;
                }
                if (!lastMerged)
                {   //incase the last one is a group as well, we need to merge it...
                    all = merge(all, groups[lastGroup], lastGroupRelation);
                    testLog.Append(")");
                }
            }
            res.Passed = all ;
            test.Result = res;
            testLog.AppendLine(" (group result: " + groups[lastGroup] + ")\r\n}\r\n )
\r\nFinal Result: " + res.Passed);
            //System.Diagnostics.Trace.WriteLine("Test: " + testLog.ToString());
            this._log.AppendLine(testLog.ToString());
            return res;
        }

        /// <summary>
        /// Evaluates two FaultDetectionParam objects based on the OperationTypeName
given
        /// </summary>
        /// <param name="collection">a collection to use for refrence when loading
parametrs or used other tests</param>
        /// <param name="one">the first FaultDetectionParam</param>
        /// <param name="two">the second FaultDetectionParam</param>
        /// <param name="operationTypeName">the operation type name to use (must have
a representing IFaultDetectionOperation implementation in the
FaultDetection.Operations namespace)</param>
        /// <returns>FaultDetectionResult based on the evaluationg of the parameters
given</returns>
        private FaultDetectionResult evaluate(ref FaultDetectionCaseCollection
collection, FaultDetectionParam one, FaultDetectionParam two, string
operationTypeName)
        {
            loadParams(ref collection, ref one, ref two);
            // if one of the paramters is null and the other is not, return false (no
operation is valid for this combination)
            if ((one.ComputedType == FaultDetectionParamType.NULL && two.ComputedType
!= FaultDetectionParamType.NULL) ||
                (one.ComputedType != FaultDetectionParamType.NULL && two.ComputedType
== FaultDetectionParamType.NULL))
            {
                return new FaultDetectionResult(false);
            }
            IFaultDetectionOperation operation = getOperation(operationTypeName);
            if (one != null && two != null)
            {
                if (operation != null)
                {
                    return operation.evaluate(one, two);
                }
                else
                {
                    return new FaultDetectionResult(false, "Invalid Operation: " +
operationTypeName + " (" + operation + ")");
```

```
                }
            }
            return new FaultDetectionResult(false, "invalid paramter 1 (" + one + "),
paramter 2 (" + two + ") and type (" + operationTypeName + ") combination");
        }

        /// <summary>
        /// Merges two boolean values based on the give FaultDetectionRelationType:
        /// <para>
        ///     <list type="bullet">
        ///     <item><description>AND = init & toJoin</description></item>
        ///     <item><description>OR = init | toJoin</description></item>
        ///     <item><description>NONE = init</description></item>
        ///     </list>
        /// </para>
        /// </summary>
        /// <param name="init">the initial boolean value</param>
        /// <param name="toJoin">the boolean value to merge/join</param>
        /// <param name="relation">the FaultDetectionRelationType to base the merge
on</param>
        /// <returns>a bool representing the merged value of the two given boolean
values</returns>
        private bool merge(bool init, bool toJoin, FaultDetectionRelationType
relation)
        {
            //Log.v(TAG,"Merging: "+init+ (relation == RELATION_OR ? " OR " : " AND ")
+ toJoin);
            switch (relation)
            {
                case FaultDetectionRelationType.OR:
                    return (init || toJoin);
                case FaultDetectionRelationType.AND:
                    return (init && toJoin);
                default:
                    return init;
            }
        }

        /// <summary>
        /// Loads parameters based on its type/value from the collection (or its
module) or the execution environment
        /// <para>
        /// Sets the parameter Compute Type and Computed Value accordingly.
        /// </para>
        /// </summary>
        /// <param name="collection">the collection (and its module) to use to load
the parameter</param>
        /// <param name="param">the FaultDetectionParam to load/set</param>
        private void loadParam(ref FaultDetectionCaseCollection collection, ref
FaultDetectionParam param)
        {
            if (param != null)
            {
                if (collection.Module != null && !String.IsNullOrEmpty(param.Value) &&
param.Type == FaultDetectionParamType.MODULE_ATTRIBUTE)
                {

param.setComputed(collection.Module.getStatusAttribute(param.Value));
                }
                else if (collection.Module != null && collection != null && param.Type
== FaultDetectionParamType.MODULE_SETTING)
                {
```

```
                    param.setComputed(collection.Module.getSetting(param.Value));
                }
                else if (_environment != null && collection != null && param.Type ==
FaultDetectionParamType.ENVIRONMENT_ATTRIBUTE)
                {
                    param.setComputed(_environment.getAttribute(param.Value));
                }
                else if (collection.Module != null && collection != null && param.Type
== FaultDetectionParamType.TEST_ID)
                {
                    int testId = param.get<int>();
                    if (testId > 0)
                    {
                        param.setComputed("false", FaultDetectionParamType.BOOLEAN);
                        foreach (FaultDetectionCase fd in collection.List)
                        {
                            if (fd.Module.TypeID == collection.Module.TypeID)
                            {
                                FaultDetectionTest test = fd.getTestById(testId);
                                if (test != null)
                                {
                                    executeTest(ref collection, ref test, true); //
executing to the test in case it was not executed already...
                                    if (test.Result != null)
                                    {

param.setComputed(test.Result.Passed.ToString(), FaultDetectionParamType.BOOLEAN);
                                    }
                                    break;
                                }
                            }
                        }
                    }
                }
                else if (param.Type == FaultDetectionParamType.FUNCTION)
                {
                    param.setComputed(executeFunction(ref collection, param.Value));
                }
            }
        }

        /// <summary>
        /// Shortcut to load two paramters at once using the loadParam method.
        /// </summary>
        /// <param name="collection">the collection (and its module) to use to load
the parameters</param>
        /// <param name="param1">the first param to load</param>
        /// <param name="param2">the second param to load</param>
        private void loadParams(ref FaultDetectionCaseCollection collection, ref
FaultDetectionParam param1, ref FaultDetectionParam param2)
        {
            loadParam(ref collection, ref param1);
            loadParam(ref collection, ref param2);
        }

        /// <summary>
        /// Shortcut to load multiple paramters at once using the loadParam method.
        /// </summary>
        /// <param name="collection">the collection (and its module) to use to load
the parameters</param>
        /// <param name="paramCollection">a collection of params to load</param>
        private void loadParams(ref FaultDetectionCaseCollection collection, ref
```

```
FaultDetectionParam[] paramCollection)
        {
            if (paramCollection != null && paramCollection.Length > 0)
            {
                for (int i = 0; i < paramCollection.Length; i++)
                {
                    loadParam(ref collection, ref paramCollection[i]);
                }
            }
        }

        /// <summary>
        /// Loads all the IFaultDetectionFunction implementations from the
FaultDetection.Functions namespace
        /// </summary>
        private void loadFunctions()
        {
            Assembly pAssembly = Assembly.GetAssembly(typeof(FaultDetectionEngine));
            if (pAssembly != null)
            {
                // searching for all the types in "FaultDetection.Functions" name
space
                foreach (Type pType in pAssembly.GetTypes().Where(t =>
String.Equals(t.Namespace, "FaultDetection.Functions",
StringComparison.Ordinal)).ToArray())
                {
                    try
                    {
                        Type pInterface =
pType.GetInterface(typeof(IFaultDetectionFunction).ToString());
                        // if this type implements the IFaultDetectionFunction
interface, it will not be null
                        if (pInterface != null)
                        {
                            // creating a new IFaultDetectionFunction object and
adding it to the Loaded Functions List.
                            ConstructorInfo pConstructor = pType.GetConstructor(new
Type[] { });
                            IFaultDetectionFunction pInstance =
pConstructor.Invoke(null) as IFaultDetectionFunction;
                            if (pInstance != null)
                            {
                                _loadedFunctions.Add(pInstance);
                            }
                        }
                    }
                    catch
                    {
                    }
                }
            }
        }

        /// <summary>
        /// Loads all the IFaultDetectionOperation implementations from the
FaultDetection.Operations namespace
        /// </summary>
        private void loadOperations()
        {
            Assembly pAssembly = Assembly.GetAssembly(typeof(FaultDetectionEngine));
            if (pAssembly != null)
            {
```

```
                // searching for all the types in "FaultDetection.Operations" name
space
                foreach (Type pType in pAssembly.GetTypes().Where(t =>
String.Equals(t.Namespace, "FaultDetection.Operations",
StringComparison.Ordinal)).ToArray())
                {
                    try
                    {
                        Type pInterface =
pType.GetInterface(typeof(IFaultDetectionOperation).ToString());
                        // if this type implements the IFaultDetectionFunction
interface, it will not be null
                        if (pInterface != null)
                        {
                            // creating a new IFaultDetectionOperation object and
adding it to the Loaded Operations List.
                            ConstructorInfo pConstructor = pType.GetConstructor(new
Type[] { });
                            IFaultDetectionOperation pInstance =
pConstructor.Invoke(null) as IFaultDetectionOperation;
                            if (pInstance != null)
                            {
                                _loadedOperations.Add(pInstance);
                            }
                        }
                    }
                    catch
                    {
                    }
                }
            }
        }

        /// <summary>
        /// Returns a string representing the given FaultDetectionParam complete
information/details.
        /// <para>
        /// Used internally for logging.
        /// </para>
        /// </summary>
        /// <param name="param">a FaultDetectionParam to parse</param>
        /// <returns>the given FaultDetectionParam infroamtion/details</returns>
        private String getParamDetails(FaultDetectionParam param)
        {
            return param.Value + " [" + param.Type + (param.Type ==
FaultDetectionParamType.MODULE_ATTRIBUTE || param.Type ==
FaultDetectionParamType.TEST_ID || param.Type == FaultDetectionParamType.FUNCTION ||
param.Type == FaultDetectionParamType.ENVIRONMENT_ATTRIBUTE ? " = " +
param.ComputedValue + " [" + param.ComputedType + "]" : "") + "]";
        }

        /// <summary>
        /// Executes a function
        /// <para>
        /// Uses the FaultDetectionCaseCollection given to load/set params.
        /// </para>
        /// </summary>
        /// <param name="collection">a refrences FaultDetectionCaseCollection to use
for loading function paramters</param>
        /// <param name="function">the full signature and param values to execute,
i.e. function_name(param,param,param)</param>
        /// <returns>FaultDetectionParam result of the function execution</returns>
```

```
        private FaultDetectionParam executeFunction(ref FaultDetectionCaseCollection
collection, string function)
        {
            // checking if any functions have been loaded...
            if (_loadedFunctions.Count > 0)
            {
                Match m = Regex.Match(function, FUNCTION_PATTERN);
                // checking if the given function string is a valid function and if
so, extracting the arugments string from it...
                if (m.Success && m.Groups.Count > 1)
                {
                    foreach (IFaultDetectionFunction func in _loadedFunctions)
                    {
                        if (func.Name.Equals(m.Groups[1].Value))
                        {
                            func.init(m.Groups[2].Value);
                            FaultDetectionParam[] fparams = func.Parameters;
                            loadParams(ref collection, ref fparams);
                            return func.execute();
                        }
                    }
                }
            }
            return new FaultDetectionParam(FaultDetectionParamType.NULL);
        }

        /// <summary>
        /// Gets an IFaultDetectionOpertion impementation from the Loaded Operations
list based on the given Operation Type Name
        /// </summary>
        /// <param name="operationTypeName">the operation type name to look
for</param>
        /// <returns>IFaultDetectionOperation representing the given operation type
name, null if not found</returns>
        private IFaultDetectionOperation getOperation(string operationTypeName)
        {
            // checking if any operation have been loaded...
            if (_loadedOperations.Count > 0)
            {
                foreach (IFaultDetectionOperation operation in _loadedOperations)
                {
                    if
(operation.GetType().Name.ToLower().Equals("faultdetectionoperation"+operationTypeName
.Replace("_","").ToLower()))
                    {
                        return operation;
                    }
                }
            }
            return null;
        }
    }
}
```

1.2.6 FaultDetectionExecution.cs

Represents a past Fault Detection Execution. Includes the FaultDetectionCaseCollection used in the execution, the time of execution and its duration.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a past Fault Detection Execution.
    /// <para>
    /// Includes the FaultDetectionCaseCollection used in the execution, the time of
execution and its duration.
    /// </para>
    /// </summary>
    public class FaultDetectionExecution
    {
        private int _id, _deleted;
        private FaultDetectionCaseCollection _faultCollection;
        private DateTime _time;
        private TimeSpan _duration;

        public int ID {
            get { return this._id; }
            set { this._id = value; }
        }
        public int Deleted
        {
            get { return this._deleted; }
            set { this._deleted = value; }
        }

        public FaultDetectionCaseCollection FaultCollection
        {
            get { return this._faultCollection; }
            set { this._faultCollection = value; }
        }

        public DateTime Time
        {
            get { return this._time; }
            set { this._time = value; }
        }

        public TimeSpan Duration
```

```
        {
            get { return this._duration; }
            set { this._duration = value; }
        }

    }
}
```

1.2.7 FaultDetectionModuleFactory.cs

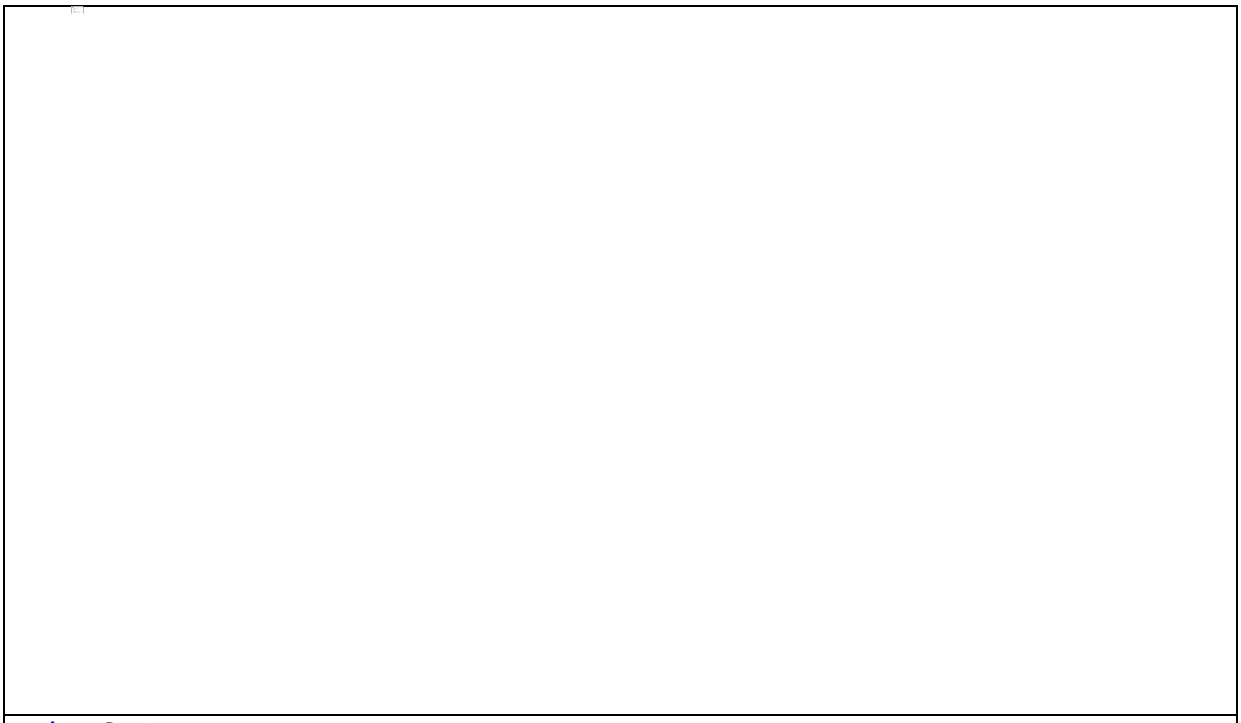
Used to create new instances of IFaultDetectionModule based an a Module Type Id or Module Type Name.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;

namespace FaultDetection
{
    /// <summary>
    /// Used to create new instances of IFaultDetectionModule based an a Module Type
Id or Module Type Name.
    /// </summary>
    public static class FaultDetectionModuleFactory
    {
        /// <summary>
        /// Creates an IFaultDetectionModule for the given Module Type Id
        /// </summary>
        /// <param name="moduleTypeId">The Type Id of the Module to create</param>
        /// <returns>IFaultDetectionModule for the give Module Type Id or null if not
found</returns>
        public static IFaultDetectionModule Create(int moduleTypeId)
        {
            return CreateModule(moduleTypeId, null);
        }

        /// <summary>
        /// Creates an IFaultDetectionModule for the given Module Type Name
        /// </summary>
        /// <param name="moduleTypeName">The Type Name of the Module to create</param>
        /// <returns>IFaultDetectionModule for the give Module Type Name or null if
not found</returns>
        public static IFaultDetectionModule Create(string moduleTypeName)
        {
```

```
            return CreateModule(-1, moduleTypeName);
        }
        /// <summary>
        /// Creates an IFaultDetectionModule for the given Module Type Id or Module
Type Name (whichever is found first)
        /// </summary>
        /// <param name="moduleTypeId">The Type Id of the Module to create</param>
        /// <param name="moduleTypeName">The Type Name of the Module to create</param>
        /// <returns>IFaultDetectionModule for the given Module Type Id or Module Type
Name (whichever is found first) or null if not found</returns>
        private static IFaultDetectionModule CreateModule(int moduleTypeId, string
moduleTypeName)
        {
            Assembly pAssembly = Assembly.GetAssembly(typeof(FaultDetectionEngine));
            if (pAssembly != null)
            {
                // searching for all the types in "FaultDetection.Modules" name space
                foreach (Type pType in pAssembly.GetTypes().Where(t =>
String.Equals(t.Namespace, "FaultDetection.Modules",
StringComparison.Ordinal)).ToArray())
                {
                    Type pInterface =
pType.GetInterface(typeof(IFaultDetectionModule).ToString());
                    // if this type implements the IFaultDetectionModule interface, it
will not be null
                    if (pInterface != null)
                    {
                        // creating a new IFaultDetectionModule object
                        ConstructorInfo pConstructor = pType.GetConstructor(new Type[]
{ });
                        IFaultDetectionModule pInstance = pConstructor.Invoke(null) as
IFaultDetectionModule;
                        if (pInstance != null)
                        {
                            // if the module type id or name match, return
                            if ((moduleTypeId > 0 && pInstance.TypeID == moduleTypeId)
|| (!string.IsNullOrEmpty(moduleTypeName) &&
pInstance.TypeName.Equals(moduleTypeName)))
                            {
                                return pInstance;
                            }
                        }
                    }
                }
            }
            return null;
        }
    }
}
```

1.2.8 FaultDetectionParam.cs

Represents different properties/attributes/characteristics of a Module, Environment, etc. Used by/in FaultDetectionTestRule to evaluate two different FaultDetectionParam to find mismatches and detect faults.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

namespace FaultDetection
{
    /// <summary>
    /// Represents different properties/attributes/characteristics of a Module,
Environment, etc.
    /// <para>
    /// Used by/in FaultDetectionTestRule to evaluate two different
FaultDetectionParam to find mismatches and detect faults
    /// </para>
    /// </summary>
    public class FaultDetectionParam
    {
        private String _value = "", _name;
        private int _id;
        private FaultDetectionParamType computedType =
FaultDetectionParamType.UNKNOWN;
        private String computedValue = "";

        public int ID
        {
            get { return this._id; }
            set { this._id = value; }
        }
        public String Value
```

```
        {
            get { return this._value; }
            set { this._value = value; }
        }
        public String Name
        {
            get { return this._name; }
            set { this._name = value; }
        }


        public String ComputedValue
        {
            get { return this.computedValue; }
        }

        [JsonConverter(typeof(StringEnumConverter))]
        public FaultDetectionParamType Type = FaultDetectionParamType.UNKNOWN;

        [JsonConverter(typeof(StringEnumConverter))]
        public FaultDetectionParamType ComputedType
        {
            get
            {
                return this.computedType;
            }
        }

        /// <summary>
        /// All classes must have a parameterless constructor so they can be
serializable.
        /// Should not be used manually.
        /// </summary>
        public FaultDetectionParam()
        {
            this.Type = FaultDetectionParamType.UNKNOWN;
            this.computedType = FaultDetectionParamType.UNKNOWN;
        }
        /// <summary>
        /// Creates a new FaultDetectionParam and tries to infer the type from the
given object and its value
        /// </summary>
        /// <param name="value">an object to use to intialise the parameter</param>
        public FaultDetectionParam(object value)
        {
            try
            {
                if (value != null)
                {
                    this.Value = value.ToString();
                    this.Type = parseType(value.GetType());
                }
                else {
                    this.Value = null;
                    this.Type = FaultDetection.FaultDetectionParamType.NULL;
                }
            }
            catch
            {
                this.Type = FaultDetectionParamType.UNKNOWN;
            }
            this.computedValue = this.Value;
```

```
            this.computedType = this.Type;
        }
        /// <summary>
        /// Creates a new FaultDetectionParam and tries to infer the type from its
value
        /// </summary>
        /// <param name="value">string representing the parameter value</param>
        public FaultDetectionParam(string value)
        {
            this.Value = value;
            this.computedValue = this.Value;
        }
        /// <summary>
        /// Creates a new FaultDetectionParam based on the given value and type
        /// </summary>
        /// <param name="value">string representing the parameter value</param>
        /// <param name="type">the FaultDetectionParamType of the parameter to
create</param>
        public FaultDetectionParam(string value, FaultDetectionParamType type)
            : this(value)
        {
            this.Type = type;
            this.computedType = this.Type;
        }
        /// <summary>
        /// Creates a new FaultDetectionParam from a knowledgebase with its id, name,
value and type.
        /// </summary>
        /// <param name="id">the id of the parameter from the knowledgebase</param>
        /// <param name="name">the name of the parameter from the
knowledgebase</param>
        /// <param name="value">string representing the parameter value</param>
        /// <param name="type">the FaultDetectionParamType of the parameter to
create</param>
        public FaultDetectionParam(int id, string name, string value,
FaultDetectionParamType type)
            : this(value, type)
        {
            this.ID = id;
            this.Name = name;
        }
        /// <summary>
        /// Creates a new FaultDetectionParam based on the given value and allows to
automatically parse the type from that value
        /// </summary>
        /// <param name="value">string representing the parameter value</param>
        /// <param name="parseTypeFromValue">whether or not to try and infer the type
of the parameter from the given value</param>
        public FaultDetectionParam(string value, bool parseTypeFromValue)
        : this(value)
        {
            if (parseTypeFromValue) {
                this.Type = getTypeFromValue(value);
                this.computedType = this.Type;
            }
        }
        /// <summary>
        /// sets the computed values of the parameter
        /// should only be used internally, from the FaultDetectionEngine.
        /// </summary>
        /// <param name="computed">the FaultDetectionParam to take the computed
information from</param>
```

```
        internal void setComputed(FaultDetectionParam computed)
        {
            if (computed != null)
            {
                setComputed(computed.Value, computed.Type);
            }
        }
        /// <summary>
        /// sets the computed values of the parameter
        /// should only be used internally, from the FaultDetectionEngine.
        /// </summary>
        /// <param name="value">the string representing the computed value</param>
        /// <param name="type">the type representing the computed type</param>
        internal void setComputed(string value, FaultDetectionParamType type)
        {
            this.computedValue = value;
            this.computedType = type;
        }
        /// <summary>
        /// Converts the FaultDetectionParam object to a type-safe object
        /// </summary>
        /// <typeparam name="T">the type to get the object as</typeparam>
        /// <returns>a type-safe object representing the FaultDetectionParam</returns>
        public T get<T>()
        {
            if (computedType == FaultDetectionParamType.UNKNOWN)
            {
                System.Diagnostics.Trace.WriteLine(computedType + " / type unknown,
parsing type");
                computedType = parseType(typeof(T));
                System.Diagnostics.Trace.WriteLine("parsed type: " + computedType);
            }
            return getFromType<T>(computedType);
        }
        /// <summary>
        /// Used internally to get the FaultDetectionParam as type-safe object based
on the given type.
        /// </summary>
        /// <typeparam name="T">the type to get the object as</typeparam>
        /// <param name="type">the FaultDetectionParamType of this parameter</param>
        /// <returns>a type-safe object representing the FaultDetectionParam</returns>
        private T getFromType<T>(FaultDetectionParamType type)
        {
            if (this.computedValue.Length > 0 && type != FaultDetectionParamType.NULL
&& type != FaultDetectionParamType.UNKNOWN)
            {
                return (T)FaultDetectionParam.parse(computedValue, type);
            }
            return default(T);
        }

        /// <summary>
        /// Parses the FaultDetectionParamType from a give type-safe Type
        /// </summary>
        /// <param name="typeOfT">the Type to try and infer the
FaultDetectionParamType from</param>
        /// <returns>a FaultDetectionParamType representing the type-safe Type
given</returns>
        private FaultDetectionParamType parseType(Type typeOfT)
        {
            if (typeOfT == null)
            {
```

```
            }
            else if (typeOfT == typeof(String))
            {
                return FaultDetectionParamType.STRING;
            }
            else if (typeOfT == typeof(String[]))
            {
                return FaultDetectionParamType.STRING_ARRAY;
            }
            else if (typeOfT == typeof(int) || typeOfT == typeof(Int32))
            {
                return FaultDetectionParamType.INTEGER;
            }
            else if (typeOfT == typeof(decimal) || typeOfT == typeof(Decimal))
            {
                return FaultDetectionParamType.DECIMAL;
            }
            else if (typeOfT == typeof(int[]) || typeOfT == typeof(Int32[]))
            {
                return FaultDetectionParamType.INT_ARRAY;
            }
            else if (typeOfT == typeof(decimal[]) || typeOfT == typeof(Decimal[]))
            {
                return FaultDetectionParamType.DECIMAL_ARRAY;
            }
            else if (typeOfT == typeof(bool) || typeOfT == typeof(Boolean))
            {
                return FaultDetectionParamType.BOOLEAN;
            }
            else if (typeOfT == typeof(DateTime))
            {
                return FaultDetectionParamType.DATE;
            }
            else if (typeOfT == typeof(TimeSpan))
            {
                return FaultDetectionParamType.TIMESPAN;
            }

            return FaultDetectionParamType.UNKNOWN;
        }
        /// <summary>
        /// Parses the given string based on the given FaultDetectionParamType
        ///
        /// Returns null if not matching FaultDetectionParamType or parsing failed.
        /// </summary>
        /// <param name="stringValue">the string to parse</param>
        /// <param name="type">the FaultDetectionParamType to base parsing on</param>
        /// <returns>a type-safe object representing the string given</returns>
        private static object parse(string stringValue, FaultDetectionParamType type)
        {
            switch (type)
            {
                case FaultDetectionParamType.REGEX_PATTERN:
                case FaultDetectionParamType.STRING:
                    return stringValue;
                case FaultDetectionParamType.STRING_ARRAY:
                    string[] sstrings = stringValue.Split(',');
                    if (sstrings == null || sstrings.Length < 1)
                    {
                        sstrings = stringValue.Split('|');
                    }
                    return sstrings;
```

```
                case FaultDetectionParamType.TEST_ID:
                case FaultDetectionParamType.INTEGER:
                    try
                    {
                        return Convert.ToInt32(stringValue);
                    }
                    catch
                    {
                        return 0;
                    }
                case FaultDetectionParamType.INT_ARRAY:
                    string[] strings = stringValue.Split(',');
                    if (strings == null || strings.Length < 1)
                    {
                        strings = stringValue.Split('|');
                    }
                    if (strings != null && strings.Length > 1)
                    {
                        int[] ints = new int[strings.Length];
                        for (int i = 0; i < strings.Length; i++)
                        {
                            try
                            {
                                ints[i] = Convert.ToInt32(strings[i]);
                            }
                            catch
                            {
                                return stringValue; // unable to parse all elements as
integers.
                            }
                        }
                        return ints;
                    }
                    return null;
                case FaultDetectionParamType.DECIMAL_ARRAY:
                    string[] dstrings = stringValue.Split(',');
                    if (dstrings == null || dstrings.Length < 1)
                    {
                        dstrings = stringValue.Split('|');
                    }
                    if (dstrings != null && dstrings.Length > 1)
                    {
                        decimal[] decs = new decimal[dstrings.Length];
                        for (int i = 0; i < dstrings.Length; i++)
                        {
                            try
                            {
                                decs[i] = Convert.ToDecimal(dstrings[i]);
                            }
                            catch
                            {
                                return stringValue; // unable to parse all elements as
decimal.
                            }
                        }
                        return decs;
                    }
                    return null;
                case FaultDetectionParamType.DECIMAL:
                    try
                    {
                        return Convert.ToDecimal(stringValue);
```

```
                    }
                    catch
                    {
                        return (decimal)0.0;
                    }
                case FaultDetectionParamType.BOOLEAN:
                    try
                    {
                        return Convert.ToBoolean(stringValue);
                    }
                    catch
                    {
                        return false;
                    }
                case FaultDetectionParamType.DATE:
                    try
                    {
                        return Convert.ToDateTime(stringValue);
                    }
                    catch
                    {
                        return new DateTime(0);
                    }
                case FaultDetectionParamType.TIMESPAN:
                    return praseTimespanString(stringValue);
            }
            return null;

        }
        /// <summary>
        /// Parses the given timespan string and returns a TimeSpan object
        ///
        /// </summary>
        /// <param name="stringValue">the timespan string to parse</param>
        /// <returns>a TimeSpan object representing the timespan string
given</returns>
        private static object praseTimespanString(string value)
        {
            try
            {
                Match m = Regex.Match(value,
@"^(\d*?)\s(milli(?:second(?:s|)|)|second(?:s|)|minute(?:s|)|hour(?:s|)|day(?:s|))$");
                if (m.Success && m.Groups.Count > 0)
                {

                    int intVal = Convert.ToInt32(m.Groups[1].Value);
                    if (intVal > 0)
                    {
                        if (m.Groups[2].Value.Equals("milli"))
                        {
                            return new TimeSpan(0, 0, (intVal / 1000));
                        }
                        else if (m.Groups[2].Value.Equals("second") ||
m.Groups[2].Value.Equals("seconds"))
                        {
                            return new TimeSpan(0, 0, intVal);
                        }
                        else if (m.Groups[2].Value.Equals("minute") ||
m.Groups[2].Value.Equals("minutes"))
                        {
                            return new TimeSpan(0, intVal, 0);
                        }
```

```
                        else if (m.Groups[2].Value.Equals("hour") ||
m.Groups[2].Value.Equals("hours"))
                        {
                            return new TimeSpan(intVal, 0, 0);
                        }
                        else if (m.Groups[2].Value.Equals("day") ||
m.Groups[2].Value.Equals("days"))
                        {
                            return new TimeSpan((intVal * 24), 0, 0);
                        }
                    }
                }
                return TimeSpan.Parse(value);
            }
            catch
            {
                return new TimeSpan(0);
            }

        }
        /// <summary>
        /// Uses regular expressions to detect the FaultDetectionParamType from a
given string value
        /// </summary>
        /// <param name="value">the string value of the param</param>
        /// <returns>FaultDetectionParamType infered from the given string
value</returns>
        private static FaultDetectionParamType getTypeFromValue(string value)
        {
            if (string.IsNullOrEmpty(value)) {
                return FaultDetectionParamType.NULL;
            }

            Match m = Regex.Match(value, @"^(True|False)$", RegexOptions.IgnoreCase);
            if (m.Success)
            {
                return FaultDetectionParamType.BOOLEAN;
            }
            m = Regex.Match(value, @"^(\d*?)$");
            if (m.Success)
            {
                return FaultDetectionParamType.INTEGER;
            }
            m = Regex.Match(value,
@"^\d{1,2}/\d{1,2}/\d{2,4}\s\d{2}:\d{2}(?::\d{2}|)$");
            if (m.Success)
            {
                return FaultDetectionParamType.DATE;
            }
            m = Regex.Match(value,
@"^(\d*?)\s(milli(?:second(?:s|)|)|second(?:s|)|minute(?:s|)|hour(?:s|)|day(?:s|))$");
            if (m.Success && m.Groups.Count > 0)
            {
                return FaultDetectionParamType.TIMESPAN;
            }
            m = Regex.Match(value, @"^(\d*?)\.(\d*?)$");
            if (m.Success)
            {
                return FaultDetectionParamType.DECIMAL;
            }
            if (value.IndexOf(',') > -1)
            {
```

```
                string[] split = value.Split(',');
                if (split != null && split.Length > 1)
                {
                    return getTypeFromValueArray(split);
                }
            }
            else if (value.IndexOf('|') > -1)
            {
                string[] split = value.Split('|');
                if (split != null && split.Length > 1)
                {
                    return getTypeFromValueArray(split);
                }
            }
            return FaultDetectionParamType.STRING;
        }

        /// <summary>
        /// Gets an array of strings and gets their FaultDetectionParamType Array
type.
        /// if not all string are either FaultDetectionParamType.INTEGER or all
FaultDetectionParamType.DECIMAL
        /// will return FaultDetectionParamType.STRING_ARRAY.
        /// </summary>
        /// <param name="strings">an array of values (strings) to parse</param>
        /// <returns>FaultDetectionParamType.INT_ARRAY if all values are integers,
FaultDetectionParamType.DECIMAL_ARRAY if all values are decimals,
FaultDetectionParamType.STRING_ARRAY otherwise.</returns>
        private static FaultDetectionParamType getTypeFromValueArray(string[] strings)
        {
            if (strings == null)
            {
                return FaultDetectionParamType.NULL;
            }
            if (strings.Length > 1) {
                FaultDetectionParamType type = getTypeFromValue(strings[0]);
                // if the 1st isnt a string/null but is an integer or decimal, check
the rest
                if ((type != FaultDetectionParamType.STRING && type !=
FaultDetectionParamType.NULL) && (type == FaultDetectionParamType.INTEGER | type ==
FaultDetectionParamType.DECIMAL))
                {
                    for (int i = 1; i < strings.Length; i++)
                    {
                        FaultDetectionParamType ntype = getTypeFromValue(strings[i]);
                        // verify all types are the same
                        if (type != ntype)
                        {
                            return FaultDetectionParamType.STRING_ARRAY;
                        }
                    }
                    // all types are the same, and are not string
                    return (type == FaultDetectionParamType.INTEGER ?
FaultDetectionParamType.INT_ARRAY : FaultDetectionParamType.DECIMAL_ARRAY);
                }
            }
            // not epty and not null, is an array but no specific type found
            return FaultDetectionParamType.STRING_ARRAY;
        }
    }

}
```

1.2.9 FaultDetectionParamType.cs

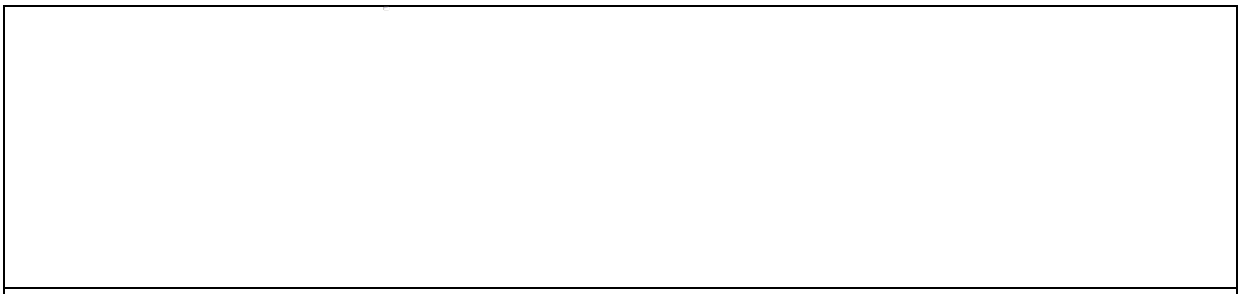
Represents the different types of FaultDetectionParams (real types of Properties/attributes/characteristics of a Module, Environment, etc.)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

namespace FaultDetection
{
    /// <summary>
    /// Represents the different types of FaultDetectionParams (real types of
properties/attributes/characteristics of a Module, Environment, etc.)
    /// </summary>
    [JsonConverter(typeof(StringEnumConverter))]
    public enum FaultDetectionParamType : int
    {
        UNKNOWN = -1,

        NULL = 0,
        STRING = 1,
        INTEGER = 2,
        DECIMAL = 3,

        STRING_ARRAY = 4,
        INT_ARRAY = 5,
        DECIMAL_ARRAY = 6,

        BOOLEAN = 7,
        DATE = 9,
        TIMESPAN = 10,
```

```
        MODULE_ATTRIBUTE = 20,
        MODULE_SETTING = 21,

        TEST_ID = 40,
        ENVIRONMENT_ATTRIBUTE = 50,

        REGEX_PATTERN = 60,

        FUNCTION = 70

    }
}
```

1.2.10 FaultDetectionRelationType.cs

Reperents the different types of Relations between FaultDetectionTestRules in a FaultDetectionTest.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

namespace FaultDetection
{
    /// <summary>
    /// Reperents the different types of Relations between FaultDetectionTestRules in
a FaultDetectionTest.
    /// </summary>
    [JsonConverter(typeof(StringEnumConverter))]
    public enum FaultDetectionRelationType : int
    {
        NONE = 0,
        AND = 1,
        OR = 2
    }
}
```

1.2.11 FaultDetectionResult.cs

Represents a FaultDetectionResult.

- Passed means fault not present
- Not Passed means fault either been detected, or check not executed
- Error is a textual reason/explanation why the check failed.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a FaultDetectionResult.
    /// <para>Passed means fault not present.</para>
    /// <para>Not Passed means fault either been detected, or check not
executed.</para>
    /// <para>Error is a textual reason/explanation why the check failed.</para>
    /// </summary>
    public class FaultDetectionResult
    {
        private bool _passed, _executed = false;
        private String _error;

        public bool Executed
        {
            get { return _executed; }
            set { this._executed = value; }
        }

        public bool Passed
        {
            get { return (_executed && _passed); }
            set { setPassed(value); }
        }

        public String Error {
            get { return (_executed ? _error : "(Not Executed)" +
(!string.IsNullOrEmpty(_error) ? " "+_error : "")); }
            set { setError(value); }
        }

        /// <summary>
        /// All classes require parameterless constructors.
        /// <para>
        /// Will create a new object with Executed = False.
```

```
        /// </para>
        /// </summary>
        public FaultDetectionResult()
        {
            this.Executed = false;
        }
        /// <summary>
        /// Creates a new FaultDetectionResult and sets it Executed to True and Passed
state.
        /// </summary>
        /// <param name="passed">Whether or not the check passed.</param>
        public FaultDetectionResult(bool passed)
        {
            setPassed(passed);

        }
        /// <summary>
        /// Creates a new FaultDetectionResult and sets it Executed to True and Passed
state.
        /// <para>
        /// Sets the Error string (even if passed).
        /// </para>
        /// </summary>
        /// <param name="passed">Whether or not the check passed.</param>
        /// <param name="error">an explanation of the fault or reason why the check
failed</param>
        public FaultDetectionResult(bool passed, String error)
            : this(passed)
        {
            this.setError(error);
        }


        /// <summary>
        /// Creates a fixed FaultDetectionResult object which is always Executed =
False, Passed = False and
        /// a fixed error description explaining the FaultDetectionParam combination
and the IFaultDetectionOperation are invalid.
        /// </summary>
        /// <param name="one">the 1st param given to the operation</param>
        /// <param name="two">the 2nd param given to the operation</param>
        /// <param name="operation">the operation attempted</param>
        /// <returns>A fixed (Executed=False, Passed=False) FaultDetectionResult
object</returns>
        public static FaultDetectionResult
invalidParameterOperator(FaultDetectionParam one, FaultDetectionParam two,
IFaultDetectionOperation operation)
        {
            FaultDetectionResult res = new
FaultDetectionResult(false,operation.GetType().Name + " Opration is invalid for Param1
(" + one.Value + " [" + one.Type + "] == " + one.ComputedValue + " [" +
one.ComputedType + "]) / Param2 (" + two.Value + " [" + two.Type + "] == " +
two.ComputedValue + " [" + two.ComputedType + "]) combination");
            res.Executed = false;
            return res;
        }

        /// <summary>
        /// Negates the test result (turns passed to failed and vice versa)
        /// </summary>
        internal void negate()
        {
```

```
            //this.setPassed(!this.Passed);
            this.Passed = !this.Passed;
            if (!string.IsNullOrEmpty(this.Error))
            {
                //setError(this.Error + " [!NEGATED]");
                this.Error += " [!Negated]";
            }
        }

        private void setError(String err)
        {
            this._error = err;
            this._executed = true;
        }

        private void setPassed(bool passed)
        {
            this._passed = passed;
            this._executed = true;
        }
    }

}
```

1.2.12 FaultDetectionTeam.cs

Represents a support team (for notifications)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a support team (for notifications)
```

```
    /// </summary>
    public class FaultDetectionTeam
    {
        private int _id;
        private string _name, _email;
        private TeamFlags _flags = TeamFlags.None;
        private List<FaultDetectionCase> _faults = new List<FaultDetectionCase>(1);

        [Flags]
        public enum TeamFlags : int
        {
            None = 0
        }

        public int ID
        {
            get { return this._id; }
            set { this._id = value; }
        }
        public string Name {
            get { return this._name; }
            set { this._name = value; }
        }
        public string Email {
            get { return this._email; }
            set { this._email = value; }
        }

        public TeamFlags Flags
        {
            get { return this._flags; }
            set { this._flags = value; }
        }

        /// <summary>
        /// Collection of faults detected which this team is 'registered' for.
        /// <para>
        /// Used for notifications of detected faults.
        /// </para>
        /// </summary>
        public List<FaultDetectionCase> Faults
        {
            get { return this._faults; }
            set { this._faults = value; }
        }

        /// <summary>
        /// All classes must have a parameterless constructor so they can be
serializable.
        /// <para>
        /// Should not be used manually.
        /// </para>
        /// </summary>
        public FaultDetectionTeam()
        {
        }

        /// <summary>
        /// Creates a new FaultDetectionTeam object representing a support team (for
notifications)
        /// </summary>
        /// <param name="id">the id of the team</param>
```

```
        /// <param name="name">the name of the team</param>
        /// <param name="email">the email address of the team (where notifications are
sent to)</param>
        /// <param name="flags">the team flags</param>
        public FaultDetectionTeam(int id, string name, string email, TeamFlags flags)
        {
            this.ID = id;
            this.Name = name;
            this.Email = email;
            this.Flags = flags;
        }



    }
}
```

1.2.13 FaultDetectionTest.cs

Represent a single FaultDetectionCase check/test. Has a list of FaultDetectionTestRule, their FaultDetectionParams, FaultDetectionOperationType (and RelationTypes) to be evaluated.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represent a single FaultDetectionCase check/test.
    /// <para>
    /// Has a list of FaultDetectionTestRule, their FaultDetectionParams,
FaultDetectionOperationType (and RelationTypes) to be evaluated.
    /// </para>
    /// </summary>
    public class FaultDetectionTest
    {
        private int _id, _caseID, _order;
        private String _name = "", _successMessage = "", _failMessage = "";
```

```
        private List<FaultDetectionTestRule> _rules = new
List<FaultDetectionTestRule>(1);// Param1, Param2;
        private FaultDetectionResult _result = new FaultDetectionResult();

        public int ID
        {
            get { return this._id; }
            set { this._id = value; }
        }
        public int CaseID
        {
            get { return this._caseID; }
            set { this._caseID = value; }
        }
        public int Order
        {
            get { return this._order; }
            set { this._order = value; }
        }

        public String Name
        {
            get { return this._name; }
            set { this._name = value; }
        }
        public String SuccessMessage {
            get { return this._successMessage; }
            set { this._successMessage = value; }
        }
        public String FailMessage
        {
            get { return this._failMessage; }
            set { this._failMessage = value; }
        }
        public List<FaultDetectionTestRule> Rules
        {
            get { return this._rules; }
            set { this._rules = value; }
        }

        public FaultDetectionResult Result
        {
            get { return this._result; }
            set { this._result = value; }
        }

        public int Groups
        {
            get { return this.getMaxGroupNumber(); }
        }

        /// <summary>
        /// All classes must have a parameterless constructor so they can be
serializable.
        /// <para>
        /// Should not be used manually.
        /// </para>
        /// </summary>
        public FaultDetectionTest()
        {
        }
```

```
        /// <summary>
        /// Creates a new FaultDetectionTest based on pre-existing information (from
the knowlege-base)
        /// </summary>
        /// <param name="id">the id of the test</param>
        /// <param name="caseId">the id of the FaultDetectionCase the test belongs
to</param>
        /// <param name="order">the order/place of the test in the
FaultDetectionCase</param>
        /// <param name="name">the name of the test</param>
        /// <param name="sucessMessage">the message to use/display when test passed
(i.e. fualt not detect)</param>
        /// <param name="failMessage">the message to user/display when the test failed
(i.e. fault detected)</param>
        public FaultDetectionTest(int id, int caseId, int order, string name, string
sucessMessage, string failMessage)
        {
            this.ID = id;
            this.CaseID = caseId;
            this.Order = order;
            this.Name = name;
            this.SuccessMessage = sucessMessage;
            this.FailMessage = failMessage;
        }

        /// <summary>
        /// Copies the test without its results to a new FaultDetectionTest.
        /// </summary>
        /// <param name="includeRules">whether or not to copy the rules as
well</param>
        /// <param name="setComputed">whether or not to copy the computed values of
the FaultDetectParams of the rules</param>
        /// <returns>a new FaultDetectionTest which is a duplicate of this test but
without the execution result</returns>
        internal FaultDetectionTest Copy(bool includeRules, bool setComputed)
        {
            FaultDetectionTest ntest = new FaultDetectionTest(this.ID, this.CaseID,
this.Order, this.Name, this.SuccessMessage, this.FailMessage);
            if (includeRules)
            {
                foreach (FaultDetectionTestRule rule in this.Rules)
                {
                    ntest.Rules.Add(rule.Copy(setComputed));
                }
            }
            return ntest;
        }

        /// <summary>
        /// Gets the Max Group Number
        /// </summary>
        /// <returns>int representing the largest group number (i.e. the last
group)</returns>
        private int getMaxGroupNumber()
        {
            int count = 0;
            if (this.Rules != null && this.Rules.Count > 0)
            {
                foreach (FaultDetectionTestRule p in this.Rules)
                {
                    if (p.Group >= count)
                    {
```

```
                        count = p.Group;
                    }
                }
            }
            count++;
            return count;
        }
    }
}
```

1.2.14 FaultDetectionTestRule.cs

Represents one FaultDetectTest Rule, i.e. and Operation Type and a set of two parameters to check.

```

```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents one FaultDetectTest Rule, i.e. and Operation Type and a set of two
parameters to check.
    /// </summary>
    public class FaultDetectionTestRule
    {

        private  FaultDetectionParam _param1, _param2;
        private int _testId, _id, _group, _order;

        private FaultDetectionRelationType _relationType;
        private string _operationTypeName;
        private bool _negate = false;

        public int Id
        {
            get { return this._id; }
            set { this._id = value; }
        }
        public int TestId
        {
            get { return this._testId; }
            set { this._testId = value; }
        }
```

```
        public int Group
        {
            get { return this._group; }
            set { this._group = value; }
        }
        public int Order
        {
            get { return this._order; }
            set { this._order = value; }
        }

        public FaultDetectionParam Param1
        {
            get { return this._param1; }
            set { this._param1 = value; }
        }
        public FaultDetectionParam Param2
        {
            get { return this._param2; }
            set { this._param2 = value; }
        }

        public FaultDetectionRelationType RelationType
        {
            get { return this._relationType; }
            set { this._relationType = value; }
        }
        public string OperationTypeName
        {
            get { return this._operationTypeName; }
            set { this._operationTypeName = value; }
        }
        public bool Negate
        {
            get { return this._negate; }
            set { this._negate = value; }
        }

        private FaultDetectionResult result;

        public FaultDetectionResult Result
        {
            get { return this.result; }
            set { this.setResult(value); }
        }

        public FaultDetectionRelationType FaultDetectionRelationType
        {
            get
            {
                return this.RelationType;
            }
            set
            {
                this.RelationType = value;
            }
        }

        /// <summary>
        /// All classes must have a parameterless constructor so they can be
serializable.
        /// <para>Should not be used manually.</para>
```

```
        /// </summary>
        public FaultDetectionTestRule()
        {
        }

        /// <summary>
        /// Creates a new FaultDetectionTestRule
        /// </summary>
        /// <param name="testId">the id of the test</param>
        /// <param name="ruleId">the id of the test rule</param>
        /// <param name="group">the group the rule is in</param>
        /// <param name="order">the order in the group</param>
        /// <param name="relationType">the relation type to the next rule</param>
        /// <param name="operationTypeName">the operation type name to use for
evaluating</param>
        /// <param name="negate">whether or not to negate the result after
evaluating</param>
        public FaultDetectionTestRule(int testId, int ruleId, int group, int order,
FaultDetectionRelationType relationType, string operationTypeName, bool negate)
        {
            this.TestId = testId;
            this.Id = ruleId;
            this.Group = group;
            this.Order = order;
            this.RelationType = relationType;
            this.OperationTypeName = operationTypeName;
            this.Negate = negate;
        }


        /// <summary>
        /// Copies this FaultDtectionTestRule to a new instance
        /// <para>
        /// Does not set the paramemeters computed values/types
        /// </para>
        /// </summary>
        /// <returns>A copied instance of this FaultDtectionTestRule
instance</returns>
        internal FaultDetectionTestRule Copy()
        {
            return this.Copy(false);
        }

        /// <summary>
        /// Copies this FaultDtectionTestRule (without its result) to a new instance
        /// </summary>
        /// <param name="includeComputed">Whether or not to set the parameters
computed values</param>
        /// <returns>A copied instance of this FaultDtectionTestRule instance (without
its execution result)</returns>
        internal FaultDetectionTestRule Copy(bool includeComputed)
        {
            FaultDetectionTestRule nparam = new FaultDetectionTestRule(this.TestId,
this.Id, this.Group, this.Order, this.RelationType, this.OperationTypeName,
this.Negate);
            nparam.Param1 = new FaultDetectionParam(this.Param1.ID, this.Param1.Name,
this.Param1.Value, this.Param1.Type);
            nparam.Param2 = new FaultDetectionParam(this.Param2.ID, this.Param2.Name,
this.Param2.Value, this.Param2.Type);
            if (includeComputed)
            {
                nparam.Param1.setComputed(this.Param1);
```

```
                nparam.Param2.setComputed(this.Param2);
            }
            return nparam;
        }

        private void setResult(FaultDetectionResult result)
        {

            this.result = result;
            if (this.Negate)
            {
                this.result.negate();
            }
        }

    }
}
```

1.2.15 FaultDetectionUtil.cs

Collection of general Utility type methods. Used to decrease duplication of code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Collection of general Utility type methods
    /// <para>Used to decrease duplication of code.</para>
    /// </summary>
    public static class FaultDetectionUtil
    {
        /// <summary>
        /// Converts a string to its byte array
        /// </summary>
        /// <param name="str">the string to convert</param>
        /// <returns>byte array based on the given string</returns>
        public static byte[] DecodeHexString(string str)
        {
            uint num = (uint)(str.Length / 2);
            byte[] buffer = new byte[num];
            int num2 = 0;
            for (int i = 0; i < num; i++)
```

```
            {
                buffer[i] = (byte)((HexToByte(str[num2]) << 4) | HexToByte(str[num2 +
1]));
                num2 += 2;
            }
            return buffer;
        }
        /// <summary>
        /// Converts a Char to its byte value.
        /// </summary>
        /// <param name="val">the char to convert</param>
        /// <returns>byte representing the given char</returns>
        public static byte HexToByte(char val)
        {
            if ((val <= '9') && (val >= '0'))
            {
                return (byte)(val - '0');
            }
            if ((val >= 'a') && (val <= 'f'))
            {
                return (byte)((val - 'a') + 10);
            }
            if ((val >= 'A') && (val <= 'F'))
            {
                return (byte)((val - 'A') + 10);
            }
            return 0xff;
        }
        /// <summary>
        /// Converts "text" (human readable) string in Hexdecimal form String back to
its "textual" meaning.
        /// <para>Used for passing data between the XML WebServices and the Engine
when </para>
        /// <para>the "textual" string might containt invalid XML characters.</para>
        /// </summary>
        /// <param name="hexString">the hexdecimal string to convert</param>
        /// <returns>a text string based on the hexdecimal string given.</returns>
        public static string HexToString(string hexString) {
            return System.Text.Encoding.Default.GetString(DecodeHexString(hexString));
        }
    }
}
```

1.3 Interfaces

1.3.1 IFaultDetectionEnvironment.cs

Represents a FaulDetection Environment

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a FaulDetection Environment
    /// </summary>
    public interface IFaultDetectionEnvironment
    {
        /// <summary>
        /// Gets one of the Environments Attributes, by its name.
        /// </summary>
        /// <param name="attributeName">the name of the attribute to get</param>
        /// <returns>FaultDetectionParam object representing the Attribute (if such
attribute exists)</returns>
        FaultDetectionParam getAttribute(String attributeName);

        /// <summary>
        /// The Enviornments Type ID
        /// </summary>
        int EnvironmentTypeID { get; }

        /// <summary>
        /// The Enviornments Type Name
        /// </summary>
        string EnvironmentTypeName { get; }
    }
}
```

1.3.2 IFaultDetectionFunction.cs

Represents a Fault Detection Function

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a Fault Detection Function
    /// </summary>
    interface IFaultDetectionFunction
    {
        /// <summary>
        /// The functions name
        /// <para>
        /// Used to detect if the implementation is for a given function name (from
the db)
        /// </para>
        /// </summary>
        string Name
        {
            get;
        }

        /// <summary>
        /// Returns the functions help
        /// <para>
        /// Will be used at a later stage by the explanation facility
        /// </para>
        /// </summary>
        string Help
        {
            get;
        }

        /// <summary>
        /// Initalises the function based on the given signature (withouth the
function name):
        /// <para>function_name(param,param,param,...) -> param,param,param,...</para>
        /// </summary>
        /// <example>
        /// <code>
        ///     init("param1,param2,param2,param2,...");
        /// </code>
        /// </example>
        /// <param name="function_arguments">the functions signature, without the name
or brackets, i.e. arguments only</param>
```

```
        /// <returns>true if initialization succeeded, false otherwise.</returns>
        bool init(string function_arguments);

        /// <summary>
        /// Gets the parameters used by the function
        /// <para>
        /// (based on the signature used in the init method)
        /// </para>
        /// </summary>
        FaultDetectionParam[] Parameters { get; }

        /// <summary>
        /// Executes the function and returns the resulting FaultDetectionParam
        /// </summary>
        /// <returns>FaultDetectionParam representing the result of the function
execution</returns>
        FaultDetectionParam execute();
    }
}
```

1.3.3 IFaultDetectionKnowledgebase.cs

Represents a Fault Detection Knowlege-base

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a Fault Detection Knowlege-base
    /// </summary>
    public interface IFaultDetectionKnowledgebase : IDisposable
    {
        /// <summary>
        /// Checks if a given connection string is for the current interface
implementation.
        /// </summary>
        /// <param name="connectionString">The knowledgebase (database) connection
string</param>
        /// <returns>true if the connection string is for the current interface
implementation, false otherwise</returns>
        bool isConnectionStringForDB(string connectionString);
```

```
        /// <summary>
        /// Gets or Sets the current implementation connection string
        /// </summary>
        string ConnectionString { get; set; }

        /// <summary>
        /// Gets a complete fault detection case collection for the given Module (by
its TypeId).
        /// </summary>
        /// <param name="module">the module the fault detection case collection is
for</param>
        /// <returns>FaultDetectionCaseCollection object representing all the checks
to execute for the given module</returns>
        FaultDetectionCaseCollection getFaultDetectionCollection(IFaultDetectionModule
module);

        /// <summary>
        /// Saves a FaultDetectionCaseCollection object after its been executed.
        /// <para>
        /// Includes saving all the tests, params and their values, as well as the
results.
        /// </para>
        /// <para>
        /// Saved executions are later used by end-users using the WebAPI/Web
Explanation Facility to review Fault Detection results.
        /// </para>
        /// </summary>
        /// <param name="fdCollection">the executed fault detection case collection to
save</param>
        /// <returns>FaultDetectionExecution object representing the saved
execution.</returns>
        FaultDetectionExecution
saveFaultDetectionExecution(FaultDetectionCaseCollection fdCollection);

        /// <summary>
        /// Gets a past FaultDetectionExecution from the Knowledgebase based on its
ID.
        /// <para>
        /// Can be filtered to only get the failed FaultDetectionCases from that
execution.
        /// </para>
        /// </summary>
        /// <param name="id">the id of the past execution to get</param>
        /// <param name="failedOnly">whether or not to only get FaultDetectionCases
which failed</param>
        /// <returns>FaultDetectionExecution object representing a past fault
detection execution.</returns>
        FaultDetectionExecution getFaultDetectionExecution(int id, bool failedOnly);
    }
}
```

1.3.4 IFaultDetectionModule.cs

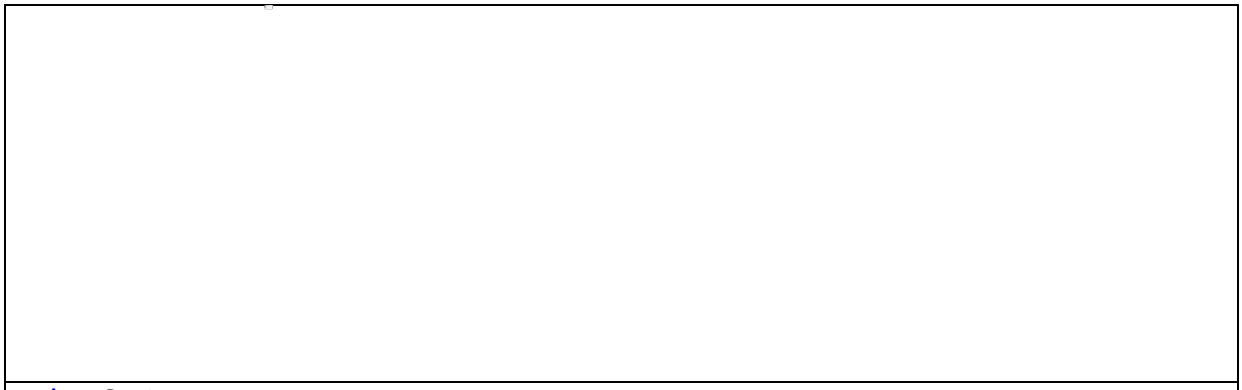
Represents a module to executed a Fault Detection on

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a module to executed a Fault Detection on
    /// </summary>
    public interface IFaultDetectionModule
    {
        /// <summary>
        /// Gets one of the Status Attributes from the module, based on the attributes
name
        /// </summary>
        /// <param name="attributeName">the name of the attribute to get</param>
        /// <returns>FaultDetectionParam representing the attribute (would be of
FaultDetectionParamType NULL if not such attribute exists)</returns>
        FaultDetectionParam getStatusAttribute(String attributeName);

        /// <summary>
        /// Finds one of the Status Atttibutes without its exact name.
        /// </summary>
        /// <param name="attributeName">the "search" string to use for finding the
attribute. Will return the first matching attribute</param>
        /// <returns>FaultDetectionParam representing the found attribute (would be of
FaultDetectionParamType NULL if no attribute was found)</returns>
        FaultDetectionParam findStatusAttribute(String searchString);

        /// <summary>
        /// Gets one of the Module settings basd on the settings name
        /// </summary>
        /// <param name="settingName">the name of the setting to get</param>
        /// <returns>FaultDetectionParam object representing the given setting
name</returns>
        FaultDetectionParam getSetting(String settingName);

        /// <summary>
        /// The modules Type Id
        /// </summary>
        int TypeID { get; }
```

```
        /// <summary>
        /// The modules Type Name
        /// </summary>
        string TypeName { get; }

        /// <summary>
        /// The modules Unique Id (can be MacAddress, IP Address, etc.)
        /// </summary>
        string UniqueId { get; set; }

        /// <summary>
        /// The modules Unique Name
        /// </summary>
        string UniqueName { get; set; }

        /// <summary>
        /// Gets or Sets the modules properties
        /// </summary>
        Dictionary<string, string> Properties { get; set; }

        /// <summary>
        /// Loads a State/Status String
        /// </summary>
        /// <param name="status">the state/status string to load</param>
        /// <returns>int representing the number of Attributes loaded</returns>
        /// <exception cref="FormatException">thrown when the give State/Status string
does not match the implemeting module format</exception>
        int loadStatus(String status);
    }
}
```

1.3.5 IFaultDetetctionNotifier.cs

Represents a Fault Detection Notifier



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a Fault Detection Notifier
    /// </summary>
    public interface IFaultDetectionNotifier
    {
        /// <summary>
        /// Loads the settings to use for the notifier
```

```
        /// </summary>
        /// <param name="settings">a Dictionary represeting the setting name and
value</param>
        void loadSettings(Dictionary<string, string> settings);

        /// <summary>
        /// The executed FaultDetectionCaseCollection which the notifier uses to
generate notifications.
        /// </summary>
        FaultDetectionCaseCollection CaseCollection { get; set; }

        /// <summary>
        /// Sends the notification
        /// </summary>
        void sendNotifications();
    }
}
```

1.3.6 IFaultDetectionOperation.cs

Represents a Fault Detection Test Rules Parameters Operation, for example "Equals" or "Larger Than".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection
{
    /// <summary>
    /// Represents a Fault Detection Test Rules Parameters Operation, for example
"Equals" or "Larger Than".
    /// </summary>
    public interface IFaultDetectionOperation
    {
        /// <summary>
        /// Evaluates two paramters based on the interface implementation.
        /// </summary>
        /// <param name="one">The first param to use in the operation</param>
        /// <param name="two">The second param to use in the operation</param>
        /// <returns>FaultDetectionResult based on the interface
implementation</returns>
        FaultDetectionResult evaluate(FaultDetectionParam one, FaultDetectionParam
two);
    }
}
```

1.4 Interfaces Implementations

1.4.1 Enviornments\FaultDetectionEnvironmentFGBFCS.cs

IFaultDetectionEnvrionment - represents a Fortress Football Club Scheme (FCS) Environment (Type ID: 1, Type Name: FGBFCS)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Environments
{
    /// <summary>
    /// Represents a Fortress Football Club Scheme (FCS) Environment
    /// <para>
    /// <list type="bullet">
    /// <item><description>Type ID: 1</description></item>
    /// <item><description>Type Name: FGBFCS</description></item>
    /// </list>
    /// </para>
    /// </summary>
    public class FaultDetectionEnvironmentFGBFCS : IFaultDetectionEnvironment
    {
        private const int _EnvironmentTypeID = 1;
        private const string _EnvironmentTypeName = "FGBFCS";

        public Dictionary<String, FaultDetectionParam> Attributes = new
Dictionary<String, FaultDetectionParam>(1);

        #region IFaultDetectionEnvironment Members

        public int EnvironmentTypeID
        {
            get { return _EnvironmentTypeID; }
        }
        public string EnvironmentTypeName
        {
            get { return _EnvironmentTypeName; }
        }

        public FaultDetectionParam getAttribute(string attributeName)
        {
            foreach (KeyValuePair<String, FaultDetectionParam> attr in Attributes)
            {
                if (attr.Key.Equals(attributeName))
                {
                    return attr.Value;
                }
            }
            return new FaultDetectionParam("",
FaultDetection.FaultDetectionParamType.NULL);
        }

        #endregion
    }
}
```

1.4.2 Functions\ FaultDetectionFunctionRegexExtract.cs

IFaultDetectionFunctions - A Regular Experssions Extract Function. Extracts a part of a string using a regular expression pattern

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace FaultDetection.Functions
{
    /// <summary>
    /// A Regular Experssions Extract Function.
    /// <para>
    /// Extracts a part of a string using a regular expression pattern
    /// </para>
    /// <list type="bullet">
    ///     <item><description>@param string ParamValue</description></item>
    ///     <item><description>@param int ParamType [Optional]</description></item>
    ///     <item><description>@param string RegexPattern</description></item>
    ///     <item><description>@param int GroupToExtract</description></item>
    /// </list>
    /// </summary>
    /// <example>
    /// Load MODULE_ATTRIBUTE called 'Module.Attribute' (actual value = '1 [NAME] at
01/01/2011') and extract group 3 from it "01/01/2011"
    /// <code>
    ///     init("Module.Attribute,20,(\d*?) \[(.*\)] at \((.*?)\),3");
    /// </code>
    /// </example>
    class FaultDetectionFunctionRegexExtract : IFaultDetectionFunction
    {
        private const string function_name = "regex_extract";

        private string _function_arguments;
        private FaultDetectionParam _param;
        private int _extractGroup = 0;
        private string _pattern;

        #region IFaultDetectionFunction Members

        public string Name
        {
            get { return function_name; }
        }

        public string Help
        {
            get {
                StringBuilder sb = new StringBuilder();
                sb.AppendLine("Extracts a part of a string using a regular expression
pattern");
                sb.AppendLine("@param string ParamValue");
                sb.AppendLine("@param int ParamType [Optional]");
                sb.AppendLine("@param string RegexPattern");
                sb.AppendLine("@param int GroupToExtract");
                sb.AppendLine("Examples:");
                sb.AppendLine(this.Name + @"(Module.Attribute,20,(\d*?) \[(.*\)] at
\((.*?)\),3) : Load MODULE_ATTRIBUTE called 'Module.Attribute' (actual value = '1
[NAME] at 01/01/2011') and extract group 1 from it (01/01/2011)");
```

```
                sb.AppendLine(this.Name + @"(123455 [Something],(\d*?) \[.*\],1) :
Create a new param with value '123455 [Something]', infer its actual type (STRING),
and extract group 1 from it (123455)");
                sb.AppendLine(this.Name + @"(123455 [Something],(\d*?) \[(.*?)\],2) :
Create a new param with value '123455 [Something]', infer its actual type (STRING),
and extract group 2 from it (Something)");
                return sb.ToString();
            }
        }

        public FaultDetectionParam[] Parameters
        {
            get { return new FaultDetectionParam[] { _param }; }
        }

        /// <summary>
        /// <para>
        /// Extracts a part of a string using a regular expression pattern
        /// </para>
        /// <list type="bullet">
        ///     <item><description>@param string ParamValue</description></item>
        ///     <item><description>@param int ParamType
[Optional]</description></item>
        ///     <item><description>@param string RegexPattern</description></item>
        ///     <item><description>@param int GroupToExtract</description></item>
        /// </list>
        /// </summary>
        /// <example>
        /// Load MODULE_ATTRIBUTE called 'Module.Attribute' (actual value = '1 [NAME]
at 01/01/2011') and extract group 3 from it "01/01/2011"
        /// <code>
        ///     init("Module.Attribute,20,(\d*?) \[(.*\)] at \((.*?)\),3");
        /// </code>
        /// </example>
        /// <param name="function_arguments">the functions argument, i.e.
(param,param,param,param)</param>
        /// <returns>true if intialization succeeded, otherwise false</returns>
        public bool init(string function_arguments)
        {
            this._function_arguments = function_arguments;
            Match m = Regex.Match(this._function_arguments,
@"^(.*?)\,(\d*?)\,(.*?)\,(\d*?)$");
            if (m.Success && m.Groups.Count > 4)
            {
                int paramType = 0;
                try
                {
                    paramType = Convert.ToInt32(m.Groups[2].Value);
                    _extractGroup = Convert.ToInt32(m.Groups[4].Value);
                }
                catch
                {
                }
                _param = new FaultDetectionParam(m.Groups[1].Value,
(FaultDetectionParamType)paramType);
                _pattern = m.Groups[3].Value;
                return true;
            }
            else
            {
                m = Regex.Match(this._function_arguments, @"^(.*?)\,(.*?)\,(\d*?)$");
                if (m.Success && m.Groups.Count > 3)
```

```
                {
                    try
                    {
                        _extractGroup = Convert.ToInt32(m.Groups[3].Value);
                    }
                    catch
                    {
                        _extractGroup = 0;
                    }
                    _param = new FaultDetectionParam(m.Groups[1].Value, true);
                    _pattern = m.Groups[2].Value;
                    return true;
                }
            }

            return false;
        }
        /// <summary>
        /// Extracts a certain part of a string based on a regular expression patten
and generates a FaulDetectionParam object from it.
        /// </summary>
        /// <returns>FaultDetectionParam representing the extracted value</returns>
        public FaultDetectionParam execute()
        {
            if (_param != null && _extractGroup > 0 &&
!string.IsNullOrEmpty(_pattern))
            {

                if (_param.ComputedType == FaultDetectionParamType.STRING)
                {
                    Match mx = Regex.Match(_param.ComputedValue, _pattern);
                    if (mx.Success && mx.Groups.Count > _extractGroup)
                    {
                        return new FaultDetectionParam(mx.Groups[_extractGroup].Value,
true);
                    }
                }
            }
            return new FaultDetectionParam(FaultDetectionParamType.NULL);
        }

        #endregion
    }
}
```

1.4.3 Knowledgebases\Sql\SqlDatabase.cs

IFaultDetectionKnowledgebase - Represents a Knowledge-base stored on a MS-SQL Database.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using FaultDetection.Environments;
using System.Text.RegularExpressions;

namespace FaultDetection.Knowledgebases.Sql
{
   /// <summary>
   /// Represents a Knowledge-base stored on a MS-SQL Database.
   /// </summary>
```

```
   public class SqlDatabase : IFaultDetectionKnowledgebase
    {
        private struct View {
            public const string Cases = "vFaultDetection_Cases";
            public const string Cases_Notifications =
"vFaultDetection_Cases_Notifications";
            public const string Tests = "vFaultDetection_Tests";
            public const string Rules = "vFaultDetection_Tests_Rules";
            public const string ModuleTypes = "FaultDetection_ModuleTypes";
            public const string Executions = "vFaultDetection_Executions";
        }
        private struct Prefix
        {
            public const string FaultDetection = "FaultDetection_";
            public const string Cases = "FaultDetection_Case_";
            public const string Tests = "FaultDetection_Test_";
            public const string Rules = "FaultDetection_Rule_";
            public const string ModuleTypes = "FaultDetection_";
            public const string ModuleProperties = "FaultDetection_Module_Property_";
            public const string Notification_Teams =
"FaultDetection_Notification_Team_";
            public const string Executions = "FaultDetection_Execution_";
        }
        private struct StoredProcedures
        {
            public const string EnvironmentGet =
"stp_WebServer_Ext_FaultDetection_Environment";
            public const string ExecutionGet =
"stp_External_FaultDetection_Execution_Get";
            public const string CollectionGet = "stp_External_FaultDetection_Get";
        }
        private string _connectionString = "";

        public string ConnectionString {
            get { return this._connectionString; }
            set { this._connectionString = value; }
        }

        /// <summary>
        /// Parameterless constructor used for loading this object using Reflection at
runtime
        /// </summary>
        public SqlDatabase()
       {

       }
        /// <summary>
        /// Creates a new SqlDatabase object with the given MS-SQL connection string
        /// </summary>
        /// <param name="connectionString">the MS-SQL connection string to use to
connect to the Database</param>
        public SqlDatabase(string connectionString)
        {
            this._connectionString = connectionString;
        }
        /// <summary>
        /// Checks if a given connection string is for the current interface
implementation.
        /// </summary>
        /// <param name="connectionString">The database connection string</param>
        /// <returns>true if the connection string is for the current interface
implementation, false otherwise</returns>
```

```
        public bool isConnectionStringForDB(string connectionString)
        {
            return Regex.IsMatch(connectionString,
@"^server=.*;database=.*;uid=.*;pwd=.*$", RegexOptions.IgnoreCase);
        }

        /// <summary>
        /// Gets a complete fault detection case collection for the given Module (by
its TypeId).
        /// </summary>
        /// <param name="module">the module the fault detection case collection is
for</param>
        /// <returns>FaultDetectionCaseCollection object representing all the checks
to execute for the given module</returns>
        public FaultDetectionCaseCollection
getFaultDetectionCollection(IFaultDetectionModule module)
        {
            FaultDetectionCaseCollection fdCollection = new
FaultDetectionCaseCollection();
            try
            {
                using (SqlExecutor Exec = new SqlExecutor(_connectionString))
                {
                    Exec.CommandType = CommandType.StoredProcedure;
                    Exec.CommandText = StoredProcedures.CollectionGet;

                    Exec.AddParam("@ModuleTypeId", SqlDbType.Int,
ParameterDirection.Input, module.TypeID);

                    using (SqlDataReader R = Exec.FetchReader())
                    {
                        if (R != null && R.HasRows)
                        {
                            fdCollection.Module = module;
                            // loading cases
                            while (R.Read())
                            {
                                fdCollection.List.Add(loadCase(module, R));
                            }
                            // loading notification teams
                            if (R.NextResult())
                            {
                                loadCollectionTestTeams(R, fdCollection);
                            }
                            // loading tests
                            if (R.NextResult())
                            {
                                loadCollectionTests(R, fdCollection);
                            }
                            // loading test rules
                            if (R.NextResult())
                            {
                                loadCollectionTestsRules(R, fdCollection);
                            }
                        }
                    }
                }
            }
            catch //(Exception ex)
            {

            }
```

```
            return fdCollection;
        }

        /// <summary>
        /// Gets a past FaultDetectionExecution from the Database based on its ID.
        /// <para>
        /// Can be filtered to only get the failed FaultDetectionCases from that
execution.
        /// </para>
        /// </summary>
        /// <param name="id">the id of the past execution to get</param>
        /// <param name="failedOnly">whether or not to only get FaultDetectionCases
which failed</param>
        /// <returns>FaultDetectionExecution object representing a past fault
detection execution.</returns>
        public FaultDetectionExecution getFaultDetectionExecution(int id, bool
failedOnly)
        {
            FaultDetectionExecution fdExecution = new FaultDetectionExecution();
            try
            {
                using (SqlExecutor Exec = new SqlExecutor(_connectionString))
                {
                    Exec.CommandType = CommandType.StoredProcedure;
                    Exec.CommandText = StoredProcedures.ExecutionGet;

                    Exec.AddParam("@ExecutionID", SqlDbType.Int,
ParameterDirection.Input, id);
                    Exec.AddParam("@FailedOnly", SqlDbType.Int,
ParameterDirection.Input, (failedOnly ? 1 : 0));

                    using (SqlDataReader R = Exec.FetchReader())
                    {
                        if (R.Read())
                        {
                            IFaultDetectionModule module =
FaultDetectionModuleFactory.Create(SqlExecutor.getInt(R, Prefix.Executions +
"Module_TypeID"));
                            if (module != null)
                            {
                                fdExecution = loadExecution(R, module);
                                if (fdExecution.ID == id)
                                {
                                    // loading module properties
                                    if (R.NextResult())
                                    {
                                        while (R.Read())
                                        {

fdExecution.FaultCollection.Module.Properties.Add(SqlExecutor.getString(R,
Prefix.ModuleProperties + "Name"), SqlExecutor.getString(R, Prefix.ModuleProperties +
"Value"));
                                        }
                                    }
                                    // loading cases
                                    if (R.NextResult())
                                    {
                                        while (R.Read())
                                        {

fdExecution.FaultCollection.List.Add(loadCase(module, R));
                                        }
```

```
                                    }
                                    // loading notification teams
                                    if (R.NextResult())
                                    {
                                        loadCollectionTestTeams(R,
fdExecution.FaultCollection);
                                    }
                                    // loading tests
                                    if (R.NextResult())
                                    {
                                        loadCollectionTests(R,
fdExecution.FaultCollection);
                                    }
                                    // loading test rules
                                    if (R.NextResult())
                                    {
                                        loadCollectionTestsRules(R,
fdExecution.FaultCollection);

                                    }
                                }
                            }
                        }
                    }
                }
            }
            catch //(Exception ex)
            {
            }

            return fdExecution;
        }

        /// <summary>
        /// Saves a FaultDetectionCaseCollection object after its been executed.
        /// <para>
        /// Includes saving all the tests, params and their values, as well as the
results.
        /// </para>
        /// <para>
        /// Saved executions are later used by end-users using the WebAPI/Web
Explanation Facility to review Fault Detection results.
        /// </para>
        /// </summary>
        /// <param name="fdCollection">the executed fault detection case collection to
save</param>
        /// <returns>FaultDetectionExecution object representing the saved
execution.</returns>
        public FaultDetectionExecution
saveFaultDetectionExecution(FaultDetectionCaseCollection fdCollection)
        {
            FaultDetectionExecution fdExecution = new FaultDetectionExecution();
            try
            {
                fdExecution.FaultCollection = fdCollection;
                fdExecution = executionSave(fdExecution);
                if (fdExecution.ID > 0)
                {
                    // execution saved
                    if
(executionSaveModuleProperties(fdExecution).ErrorCode.Equals("S"))
                    {
```

```
                        // all module properties saved, saving all cases:
                        for (int ic = 0; ic < fdCollection.List.Count; ic++)
                        {
                            fdCollection.List[ic] = executionSaveCase(fdExecution,
fdCollection.List[ic]);
                            if (fdCollection.List[ic] != null &&
fdCollection.List[ic].ID > 0)
                            {
                                // case saved, saving all  tests
                                for (int it = 0; it <
fdCollection.List[ic].Tests.Count; it++)
                                {
                                    fdCollection.List[ic].Tests[it] =
executionSaveTest(fdExecution, fdCollection.List[ic].Tests[it]);
                                    if (fdCollection.List[ic].Tests[it] != null &&
fdCollection.List[ic].Tests[it].ID > 0)
                                    {
                                        // test saved, saving all rules, their
parameter computed values and results
                                        for (int ip = 0; ip <
fdCollection.List[ic].Tests[it].Rules.Count; ip++)
                                        {
                                            fdCollection.List[ic].Tests[it].Rules[ip]
= executionSaveTestRules(fdExecution, fdCollection.List[ic].Tests[it].Rules[ip]);
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                catch //(Exception ex)
                {
                }
                return fdExecution;
            }


            /// <summary>
            /// Loads a Collection Test Teams.
            /// </summary>
            /// <param name="R">the SqlDataReader to load the teams from</param>
            /// <param name="fdCollection">the fault detection case collection to load the
teams to</param>
            private void loadCollectionTestTeams(SqlDataReader R,
FaultDetectionCaseCollection fdCollection) {
                if (R != null)
                {
                    while (R.Read())
                    {
                        FaultDetectionCaseTeam team = loadCaseTeam(R);
                        if (team != null && team.ID > 0)
                        {
                            FaultDetectionTeam exteam =
fdCollection.getNotificationTeamById(team.ID);
                            if (exteam == null)
                            {
                                FaultDetectionTeam nteam = new FaultDetectionTeam();
                                nteam.ID = team.ID;
                                nteam.Name = team.Name;
                                nteam.Email = team.Email;
```

```
                            nteam.Flags = team.Flags;
                            fdCollection.NotificationTeams.Add(nteam);
                        }

                        foreach (FaultDetectionCase fdCase in fdCollection.List)
                        {
                            if (fdCase.ID == team.CaseID)
                            {
                                fdCase.NotificationTeams.Add(team);
                                break;
                            }
                        }
                    }
                }
            }
        }

        /// <summary>
        /// Loads a Collection Tests.
        /// </summary>
        /// <param name="R">the SqlDataReader to load the tests from</param>
        /// <param name="fdCollection">the fault detection case collection to load the
tests to</param>
        private void loadCollectionTests(SqlDataReader R, FaultDetectionCaseCollection
fdCollection)
        {
            if (R != null)
            {
                while (R.Read())
                {
                    FaultDetectionTest fdTest = loadTest(R);
                    foreach (FaultDetectionCase fdCase in fdCollection.List)
                    {
                        if (fdCase.ID == fdTest.CaseID)
                        {
                            fdCase.Tests.Add(fdTest);
                        }
                    }
                }
            }
        }
        /// <summary>
        /// Loads a Collection Tests Rules.
        /// </summary>
        /// <param name="R">the SqlDataReader to load the tests rules from</param>
        /// <param name="fdCollection">the fault detection case collection to load the
tests rules to</param>
        private void loadCollectionTestsRules(SqlDataReader R,
FaultDetectionCaseCollection fdCollection)
        {
            if (R != null)
            {
                while (R.Read())
                {
                    FaultDetectionTestRule fdTestRule = loadRule(R);
                    foreach (FaultDetectionCase fdCase in fdCollection.List)
                    {
                        foreach (FaultDetectionTest fdTest in fdCase.Tests)
                        {
                            if (fdTest.ID == fdTestRule.TestId)
                            {
                                fdTest.Rules.Add(fdTestRule);
```

```
                            }
                        }
                    }
                }
            }
        }
        /// <summary>
        /// Loads a past fault detection check execution from the database and sets
its module to the give module
        /// </summary>
        /// <param name="R">the SqlDataReader to load the execution from</param>
        /// <param name="module">the IFaultDetectionModule to for loading the
execution module details to</param>
        private FaultDetectionExecution loadExecution(SqlDataReader R,
IFaultDetectionModule module)
        {
            FaultDetectionExecution fdExecution = new FaultDetectionExecution();
            fdExecution.ID = SqlExecutor.getInt(R,Prefix.Executions+"ID");
            fdExecution.Time = SqlExecutor.getDateTime(R,Prefix.Executions+"Time");
            fdExecution.Duration = new
TimeSpan((long)SqlExecutor.getInt(R,Prefix.Executions+"Duration")*10000000); //
converting seconds back to ticks...
            fdExecution.Deleted = SqlExecutor.getInt(R,Prefix.Executions+"Deleted");
            if (fdExecution.FaultCollection == null) {
                fdExecution.FaultCollection = new FaultDetectionCaseCollection();
            }
            fdExecution.FaultCollection.Module = module;
            fdExecution.FaultCollection.Module.UniqueId = SqlExecutor.getString(R,
Prefix.Executions + "Module_UniqueId");
            fdExecution.FaultCollection.Module.UniqueName = SqlExecutor.getString(R,
Prefix.Executions + "Module_UniqueName");
            if (fdExecution.FaultCollection.Result == null) {
                fdExecution.FaultCollection.Result = new
FaultDetectionResult((SqlExecutor.getInt(R,Prefix.Executions+"Result") !=
1),SqlExecutor.getString(R,Prefix.Executions+"Error"));
            }
            return fdExecution;
        }

        /// <summary>
        /// Loads a FaultDetectionCase from SqlDataReader and sets its module to the
give module
        /// </summary>
        /// <param name="module">the module to use for the fault case</param>
        /// <param name="R">the SqlDataReader to load the fault case from</param>
        /// <returns>FaultDetectionCase loaded from the db with the given
module</returns>
        private FaultDetectionCase loadCase(IFaultDetectionModule module,
SqlDataReader R)
        {
            FaultDetectionCase fdCase = new FaultDetectionCase(module);
            fdCase.ID = SqlExecutor.getInt(R, Prefix.Cases + "ID");
            fdCase.Order = SqlExecutor.getInt(R, Prefix.Cases + "Order");
            fdCase.Deleted = SqlExecutor.getInt(R, Prefix.Cases + "Deleted");
            fdCase.Name = SqlExecutor.getString(R, Prefix.Cases + "Name");
            fdCase.Description = SqlExecutor.getString(R, Prefix.Cases +
"Description");
            if (SqlExecutor.hasColumn(R,Prefix.Executions+"Case_Result")) {
                fdCase.Result = new
FaultDetectionResult((SqlExecutor.getInt(R,Prefix.Executions+"Case_Result") !=
1),SqlExecutor.getString(R,Prefix.Executions+"Case_Error"));
            }
```

```
            try
            {
                fdCase.Flags =
(FaultDetectionCase.FaultDetectionCaseFlags)SqlExecutor.getInt(R, Prefix.Cases +
"Flags");
            }
            catch
            {
                fdCase.Flags = FaultDetectionCase.FaultDetectionCaseFlags.NONE;
            }
            return fdCase;
        }
        /// <summary>
        /// Loads a FaultDetectionCaseTeam from a SqlDataReader
        /// </summary>
        /// <param name="R">SqlDataReader to load the team from</param>
        /// <returns>FaultDetectionCaseTeam loaded from the given
SqlDataReader</returns>
        private FaultDetectionCaseTeam loadCaseTeam(SqlDataReader R)
        {
            FaultDetectionCaseTeam team = new FaultDetectionCaseTeam();
            team.ID = SqlExecutor.getInt(R, Prefix.Notification_Teams + "ID");
            team.Name = SqlExecutor.getString(R, Prefix.Notification_Teams+ "Name");
            team.Email = SqlExecutor.getString(R,Prefix.Notification_Teams+"Email");
            team.Flags =
(FaultDetection.FaultDetectionTeam.TeamFlags)SqlExecutor.getInt(R,
Prefix.Notification_Teams + "Flags");
            team.CaseID = SqlExecutor.getInt(R, Prefix.Cases + "ID");
            return team;
        }
        /// <summary>
        /// Loads a FaultDetectionTest from a SqlDataReader
        /// </summary>
        /// <param name="R">SqlDataReader to load the test from </param>
        /// <returns>FaultDetectionTest loaded from the given SqlDataReader</returns>
        private FaultDetectionTest loadTest(SqlDataReader R)
        {
            FaultDetectionTest test = new FaultDetectionTest();
            test.CaseID = SqlExecutor.getInt(R, Prefix.Cases + "ID");
            test.ID = SqlExecutor.getInt(R, Prefix.Tests + "ID");
            test.Order = SqlExecutor.getInt(R, Prefix.Tests + "Order");
            if (test.ID == 0)
            {
                test.ID = SqlExecutor.getInt(R, Prefix.Executions + "Test_ID");
                test.CaseID = SqlExecutor.getInt(R, Prefix.Executions + "Case_ID");
                test.Order = SqlExecutor.getInt(R, Prefix.Executions + "Test_Order");
            }
            test.SuccessMessage = SqlExecutor.getString(R, Prefix.Tests +
"SuccessMessage");
            test.FailMessage = SqlExecutor.getString(R, Prefix.Tests + "FailMessage");
            test.Name = SqlExecutor.getString(R, Prefix.Tests + "Name");
            if (SqlExecutor.hasColumn(R,Prefix.Executions+"Test_Result")) {
                test.Result = new
FaultDetectionResult((SqlExecutor.getInt(R,Prefix.Executions+"Test_Result") !=
1),SqlExecutor.getString(R,Prefix.Executions+"Test_Error"));
            }

            return test;
        }
        /// <summary>
        /// Loads a FaultDetectionTestRule from a SqlDataReader
        /// </summary>
```

```
        /// <param name="R">SqlDataReader to load the test from </param>
        /// <returns>FaultDetectionTestRule loaded from the given
SqlDataReader</returns>
        private FaultDetectionTestRule loadRule(SqlDataReader R)
        {
            FaultDetectionTestRule p = new FaultDetectionTestRule();
            p.Param1 = loadParam(R, "1");
            p.Param2 = loadParam(R, "2");
            p.OperationTypeName = SqlExecutor.getString(R, Prefix.FaultDetection +
"OperationTypeName");
            p.RelationType = (FaultDetectionRelationType)SqlExecutor.getInt(R,
Prefix.Rules + "RelationType");
            p.Id = SqlExecutor.getInt(R, Prefix.Rules + "ID");
            p.TestId = SqlExecutor.getInt(R, Prefix.Tests + "ID");
            p.Order = SqlExecutor.getInt(R, Prefix.Rules + "Order");
            p.Group = SqlExecutor.getInt(R, Prefix.Rules + "Group");
            p.Negate = (SqlExecutor.getInt(R, Prefix.Rules + "Negate") == 1 ? true :
false);
            if (SqlExecutor.hasColumn(R,Prefix.Executions+"Test_Rule_Result")) {
                p.Result = new FaultDetectionResult((SqlExecutor.getInt(R,
Prefix.Executions + "Test_Rule_Result") != 1), SqlExecutor.getString(R,
Prefix.Executions + "Test_Rule_Error"));
            }
            return p;
        }
        /// <summary>
        /// Loads a single FaultDetectionParam from a SqlDataReader, based on the
param name (i.e. 1 or 2)
        /// </summary>
        /// <param name="R">SqlDataReader to load the test from </param>
        /// <returns>FaultDetectionParam loaded from the given SqlDataReader</returns>
        private FaultDetectionParam loadParam(SqlDataReader R, String Name)
        {
            FaultDetectionParamType paramType;
            try
            {
                paramType = (FaultDetectionParamType)SqlExecutor.getInt(R,
Prefix.Rules + "Param" + Name + "_Type");
            }
            catch
            {
                paramType = FaultDetectionParamType.UNKNOWN;
            }
            FaultDetectionParam fdParam = new
FaultDetectionParam(SqlExecutor.getInt(R, Prefix.Rules + "Param" + Name + "_ID"),
SqlExecutor.getString(R, Prefix.Rules + "Param" + Name), SqlExecutor.getString(R,
Prefix.Rules + "Param" + Name + "_Value"), paramType);
            if (SqlExecutor.hasColumn(R,Prefix.Executions+"Param" + Name +
"_ComputedValue")) {
                fdParam.setComputed(SqlExecutor.getString(R, Prefix.Executions +
"Param" + Name + "_ComputedValue"), (FaultDetectionParamType)SqlExecutor.getInt(R,
Prefix.Executions + "Param" + Name + "_ComputedType"));
            }
            return fdParam;
        }
        /// <summary>
        /// Loads a FaultDetectionTeam from a SqlDataReader
        /// </summary>
        /// <param name="R">SqlDataReader to load the team from</param>
        /// <returns>FaultDetectionTeam loaded from the given SqlDataReader</returns>
        private FaultDetectionTeam loadNotificationTeam(SqlDataReader R)
        {
```

```
            FaultDetectionTeam team = new FaultDetectionTeam();
            team.ID = SqlExecutor.getInt(R, Prefix.Notification_Teams + "ID");
            team.Name = SqlExecutor.getString(R, Prefix.Notification_Teams + "Name");
            team.Email = SqlExecutor.getString(R, Prefix.Notification_Teams +
"Email");
            team.Flags =
(FaultDetection.FaultDetectionTeam.TeamFlags)SqlExecutor.getInt(R,
Prefix.Notification_Teams + "Flags");
            return team;
        }

        /// <summary>
        /// Saves or Updates (based on Execution Id) a FaultDetectionExecution to the
Database
        /// </summary>
        /// <param name="fdExecution">FaultDetectionExecution to save</param>
        /// <returns>the saved FaultDetectionExecution</returns>
        private FaultDetectionExecution executionSave(FaultDetectionExecution
fdExecution)
        {
            using (SqlExecutor Exec = new SqlExecutor(_connectionString))
            {
                Exec.CommandType = CommandType.StoredProcedure;
                Exec.CommandText = "stp_External_FaultDetection_Execution_Save";

                Exec.AddParam("@ExecutionID", SqlDbType.Int, ParameterDirection.Input,
fdExecution.ID);
                Exec.AddParam("@ExecutionModuleTypeId", SqlDbType.Int,
ParameterDirection.Input, fdExecution.FaultCollection.Module.TypeID);
                Exec.AddParam("@ExecutionModuleUniqueId", SqlDbType.Text,
ParameterDirection.Input, fdExecution.FaultCollection.Module.UniqueId);
                Exec.AddParam("@ExecutionModuleUniqueName", SqlDbType.Text,
ParameterDirection.Input, fdExecution.FaultCollection.Module.UniqueName);
                Exec.AddParam("@ExecutionTime", SqlDbType.DateTime,
ParameterDirection.Input, (fdExecution.Time.Ticks > 0 ? fdExecution.Time.ToString() :
DateTime.Now.ToString()));
                Exec.AddParam("@ExecutionDuration", SqlDbType.Int,
ParameterDirection.Input,
(int)fdExecution.FaultCollection.ExecutionDuration.TotalMilliseconds);
                Exec.AddParam("@ExecutionResult", SqlDbType.Int,
ParameterDirection.Input, (fdExecution.FaultCollection.Result.Passed ? 0 : 1));
                if (!String.IsNullOrEmpty(fdExecution.FaultCollection.Result.Error))
                {
                    Exec.AddParam("@ExecutionError", SqlDbType.Text,
ParameterDirection.Input, fdExecution.FaultCollection.Result.Error);
                }
                Exec.AddParam("@Deleted", SqlDbType.Int, ParameterDirection.Input,
fdExecution.Deleted);
                using (SqlDataReader R = Exec.FetchReader())
                {
                    if (R.Read())
                    {
                        SqlResult result = new SqlResult(R);
                        if (!result.ErrorCode.Equals("S"))
                        {
                            fdExecution.ID = -1;

                        }
                        else
                        {
                            fdExecution.ID = SqlExecutor.getInt(R, Prefix.Executions +
"ID");
```

```
                            fdExecution.Time = SqlExecutor.getDateTime(R,
Prefix.Executions + "Time");
                        }
                    }
                }
            }

            return fdExecution;
        }
        /// <summary>
        /// Saves all the module properties for a FaultDetectionExecution Module (at
time of fault detection check execution)
        /// </summary>
        /// <param name="fdExecution">FaultDetectionExecution to save the module
from</param>
        /// <returns>SqlResult representing the result of the save stp</returns>
        private SqlResult executionSaveModuleProperties(FaultDetectionExecution
fdExecution)
        {
            SqlResult result = new SqlResult();
            if (fdExecution.FaultCollection.Module.Properties.Count > 0)
            {
                foreach (KeyValuePair<string, string> prop in
fdExecution.FaultCollection.Module.Properties)
                {
                    using (SqlExecutor Exec = new SqlExecutor(_connectionString))
                    {
                        Exec.CommandType = CommandType.StoredProcedure;
                        Exec.CommandText =
"stp_External_FaultDetection_Execution_ModuleProperty_Save";

                        Exec.AddParam("@ExecutionID", SqlDbType.Int,
ParameterDirection.Input, fdExecution.ID);
                        Exec.AddParam("@PropertyName", SqlDbType.Text,
ParameterDirection.Input, prop.Key);
                        Exec.AddParam("@PropertyValue", SqlDbType.Text,
ParameterDirection.Input, prop.Value);
                        using (SqlDataReader R = Exec.FetchReader())
                        {
                            if (R.Read())
                            {
                                result.loadFromRecord(R);
                                if (result == null || !result.ErrorCode.Equals("S"))
                                {
                                    break;
                                }
                            }
                        }
                    }
                }
            }
            else
            {
                result.ErrorCode = "S";
                result.ErrorNumber = 0;
            }
            return result;
        }
        /// <summary>
        /// Saves a single executed fault case from a FaultDetectionExecution
        /// </summary>
        /// <param name="fdExecution">a saved FaultDetectionExecution
```

```
execution</param>
        /// <param name="fdCase">FaultDetectionCase to add/update for the given
execution</param>
        /// <returns>a saved FaultDetectionCase</returns>
        private FaultDetectionCase executionSaveCase(FaultDetectionExecution
fdExecution, FaultDetectionCase fdCase)
        {
            using (SqlExecutor Exec = new SqlExecutor(_connectionString))
            {
                Exec.CommandType = CommandType.StoredProcedure;
                Exec.CommandText = "stp_External_FaultDetection_Execution_Case_Save";

                Exec.AddParam("@ExecutionID", SqlDbType.Int, ParameterDirection.Input,
fdExecution.ID);
                Exec.AddParam("@CaseID", SqlDbType.Int, ParameterDirection.Input,
fdCase.ID);
                Exec.AddParam("@CaseResult", SqlDbType.Int, ParameterDirection.Input,
(fdCase.Result.Passed ? 0 : 1));
                Exec.AddParam("@CaseError", SqlDbType.Text, ParameterDirection.Input,
fdCase.Result.Error);
                using (SqlDataReader R = Exec.FetchReader())
                {
                    while (R.Read())
                    {
                        SqlResult result = new SqlResult(R);
                        if (!result.ErrorCode.Equals("S"))
                        {
                            fdCase.ID = -1;
                        }
                    }
                }
            }
            return fdCase;
        }
        /// <summary>
        /// Saves a single executed fault case test from a FaultDetectionExecution
        /// </summary>
        /// <param name="fdExecution">a saved FaultDetectionExecution
execution</param>
        /// <param name="fdTest">FaultDetectionTest to add/update for the given
execution</param>
        /// <returns>a saved FaultDetectionTest</returns>
        private FaultDetectionTest executionSaveTest(FaultDetectionExecution
fdExecution, FaultDetectionTest fdTest)
        {
            using (SqlExecutor Exec = new SqlExecutor(_connectionString))
            {
                Exec.CommandType = CommandType.StoredProcedure;
                Exec.CommandText = "stp_External_FaultDetection_Execution_Test_Save";

                Exec.AddParam("@ExecutionID", SqlDbType.Int, ParameterDirection.Input,
fdExecution.ID);
                Exec.AddParam("@CaseID", SqlDbType.Int, ParameterDirection.Input,
fdTest.CaseID);
                Exec.AddParam("@TestOrder", SqlDbType.Int, ParameterDirection.Input,
fdTest.Order);
                Exec.AddParam("@TestID", SqlDbType.Int, ParameterDirection.Input,
fdTest.ID);
                Exec.AddParam("@TestResult", SqlDbType.Int, ParameterDirection.Input,
(fdTest.Result.Passed ? 0 : 1));
                if (!string.IsNullOrEmpty(fdTest.Result.Error)) {
                    Exec.AddParam("@TestError", SqlDbType.Text,
```

```
ParameterDirection.Input, fdTest.Result.Error);
                }
                using (SqlDataReader R = Exec.FetchReader())
                {
                    while (R.Read())
                    {
                        SqlResult result = new SqlResult(R);
                        if (!result.ErrorCode.Equals("S"))
                        {
                            fdTest.ID = -1;
                        }
                    }
                }
            }
            return fdTest;
        }
        /// <summary>
        /// Saves a single executed fault case test rule from a
FaultDetectionExecution
        /// </summary>
        /// <param name="fdExecution">a saved FaultDetectionExecution
execution</param>
        /// <param name="fdTest">FaultDetectionTestRule to add/update for the given
execution</param>
        /// <returns>a saved FaultDetectionTestRule</returns>
        private FaultDetectionTestRule executionSaveTestRules(FaultDetectionExecution
fdExecution, FaultDetectionTestRule fdTestRule)
        {
            using (SqlExecutor Exec = new SqlExecutor(_connectionString))
            {
                Exec.CommandType = CommandType.StoredProcedure;
                Exec.CommandText =
"stp_External_FaultDetection_Execution_Test_Rule_Save";

                Exec.AddParam("@ExecutionID", SqlDbType.Int, ParameterDirection.Input,
fdExecution.ID);
                Exec.AddParam("@RuleID", SqlDbType.Int, ParameterDirection.Input,
fdTestRule.Id);
                Exec.AddParam("@TestID", SqlDbType.Int, ParameterDirection.Input,
fdTestRule.TestId);
                Exec.AddParam("@Param1ComputedType", SqlDbType.Int,
ParameterDirection.Input, (int)fdTestRule.Param1.ComputedType);
                Exec.AddParam("@Param1ComputedValue", SqlDbType.Text,
ParameterDirection.Input, fdTestRule.Param1.ComputedValue);
                Exec.AddParam("@Param2ComputedType", SqlDbType.Int,
ParameterDirection.Input, (int)fdTestRule.Param2.ComputedType);
                Exec.AddParam("@Param2ComputedValue", SqlDbType.Text,
ParameterDirection.Input, fdTestRule.Param2.ComputedValue);
                Exec.AddParam("@RuleResult", SqlDbType.Int, ParameterDirection.Input,
(fdTestRule.Result.Passed ? 0 : 1));
                Exec.AddParam("@RuleError", SqlDbType.Text, ParameterDirection.Input,
fdTestRule.Result.Error);
                using (SqlDataReader R = Exec.FetchReader())
                {
                    while (R.Read())
                    {
                        SqlResult result = new SqlResult(R);
                        if (!result.ErrorCode.Equals("S"))
                        {
                            fdTestRule.Id = -1;
                        }
                    }
```

```
                }
            }
            return fdTestRule;

        }

        #region IDisposable Members

        public void Dispose()
        {
            //throw new NotImplementedException();
        }

        #endregion
    }
}
```

1.4.3.1 Knowledgebases\ Sql\SqlExecutor.cs

A Sql Executing Helper Class used by the SqlDatabase to execute calls to the database.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;
using System.Diagnostics;
using System.Xml;

namespace FaultDetection.Knowledgebases.Sql
{
    /// <summary>
    /// A Sql Executing Helper Class used by the SqlDatabase to execute calls to the
database.
    /// </summary>
    public class SqlExecutor : IDisposable
    {
        SqlConnection _Connection = new SqlConnection();
        SqlCommand _Command = new SqlCommand();

        /// <summary>
        /// Initialises a SqlExecutor with a connection string.
        /// </summary>
        /// <param name="conn">the MS-SQL connection string to use</param>
        public SqlExecutor(string connectionString)
        {
            _Connection.ConnectionString = connectionString;
            _Command.CommandType = CommandType.Text;
            _Command.Connection = _Connection;
            if (_Command.Connection.State == ConnectionState.Closed ||
_Command.Connection.State == ConnectionState.Broken)
            {
                _Command.Connection.Open();
            }
        }

        public string CommandText
        {
            get { return this._Command.CommandText; }
            set { this._Command.CommandText = value; }
        }
```

```
        public CommandType CommandType
        {
            get { return this._Command.CommandType; }
            set { this._Command.CommandType = value; }
        }
        /// <summary>
        /// Adds a parameters to the Store Procedure (stp) exection
        /// </summary>
        /// <param name="name">the stp param name</param>
        /// <param name="type">the stp param type</param>
        /// <param name="dir">the stp param direction (In/Out)</param>
        /// <param name="val">the stp param value</param>
        public void AddParam(string name, SqlDbType type, ParameterDirection dir,
string val)
        {
            SqlParameter param = new SqlParameter(name, type);
            param.Direction = dir;
            param.Value = val;
            _Command.Parameters.Add(param);
        }
        /// <summary>
        /// Adds a parameters to the Store Procedure (stp) exection
        /// </summary>
        /// <param name="name">the stp param name</param>
        /// <param name="type">the stp param type</param>
        /// <param name="dir">the stp param direction (In/Out)</param>
        /// <param name="val">the stp param value</param>
        public void AddParam(string name, SqlDbType type, ParameterDirection dir, int
val)
        {
            SqlParameter param = new SqlParameter(name, type);
            param.Direction = dir;
            param.Value = val;
            if (dir != ParameterDirection.Output)
            {
                _Command.Parameters.Add(param);
            }
        }
        /// <summary>
        /// Executes the store procedure (stp) and returns a SqlDataReader object with
the results
        /// </summary>
        /// <returns>SqlDataReader wit the results of the stp execution</returns>
        public SqlDataReader FetchReader()
        {
            try
            {
                SqlDataReader SDA = _Command.ExecuteReader();
                return SDA;
            }
            catch //(Exception ex)
            {
                return null;
            }
        }
        /// <summary>
        /// Executes a SQL query
        /// </summary>
        /// <param name="SQL">a SQL query to execute</param>
        /// <returns>SqlDataReader wit the results of the query</returns>
        /// <exception cref="SqlException">thrown when the SQL execution
failed</exception>
```

```
        /// <exception cref="Exception">thrown an Exception happens in an underlying
operation</exception>
        public SqlDataReader Execute(string SQL)
        {
            _Command.CommandText = SQL;
            SqlDataReader SDA = null;
            try
            {
                SDA = _Command.ExecuteReader();
            }
            catch (SqlException sqlEx)
            {
                throw sqlEx;
            }

            catch (Exception ex)
            {
                throw ex;
            }
            return SDA;

        }
        /// <summary>
        /// gets a column from a SqlDataReader object based on the name.
        /// </summary>
        /// <param name="R">SqlDataReader object</param>
        /// <param name="Column">the colum name to get</param>
        /// <returns>object representing the column name, null if not found</returns>
        private static object getColumn(SqlDataReader R, String Column)
        {
            for (int i = 0; i < R.FieldCount; i++)
            {
                if (R.GetName(i).Equals(Column,
StringComparison.InvariantCultureIgnoreCase))
                    return R[i];
            }
            return null;
        }
        /// <summary>
        /// Checks whether or not a SqlDataReader has a specific column, based on the
column name.
        /// </summary>
        /// <param name="R">SqlDataReader to search in</param>
        /// <param name="Column">the column name</param>
        /// <returns>true if SqlDataReader has such a column, false
otherwise</returns>
        public static bool hasColumn(SqlDataReader R, String Column) {
            return (getColumn(R, Column) != null);
        }
        /// <summary>
        /// Gets a value from a SqlDataReader as Integer
        /// </summary>
        /// <param name="R">SqlDataReader to get the value from</param>
        /// <param name="Column">the column name</param>
        /// <returns>int representing the value of the column from the given
SqlDataReader</returns>
        public static int getInt(SqlDataReader R, String Column)
        {
            try
            {
                object col = getColumn(R, Column);
                return (col != null && col.GetType() != typeof(DBNull) ?
```

```
Convert.ToInt32(col) : 0);
            }
            catch
            {
                return 0;
            }
        }
        /// <summary>
        /// Gets a value from a SqlDataReader as String
        /// </summary>
        /// <param name="R">SqlDataReader to get the value from</param>
        /// <param name="Column">the column name</param>
        /// <returns>String representing the value of the column from the given
SqlDataReader</returns>
        public static String getString(SqlDataReader R, String Column)
        {
            try
            {
                object col = getColumn(R, Column);
                return (col != null && col.GetType() != typeof(DBNull) ?
col.ToString() : "");
            }
            catch
            {
                return "";
            }
        }
        /// <summary>
        /// Gets a value from a SqlDataReader as DateTime
        /// </summary>
        /// <param name="R">SqlDataReader to get the value from</param>
        /// <param name="Column">the column name</param>
        /// <returns>DateTime representing the value of the column from the given
SqlDataReader</returns>
        public static DateTime getDateTime(SqlDataReader R, String Column)
        {
            try
            {
                object col = getColumn(R, Column);
                if (col != null && col.GetType() != typeof(DBNull))
                {
                    return DateTime.Parse(col.ToString());
                }
            }
            catch
            {
                System.Diagnostics.Trace.WriteLine("Failed getting (DateTime): " +
Column);
            }
            return new DateTime(0);
        }
        /// <summary>
        /// Gets a value from a SqlDataReader as Double
        /// </summary>
        /// <param name="R">SqlDataReader to get the value from</param>
        /// <param name="Column">the column name</param>
        /// <returns>Double representing the value of the column from the given
SqlDataReader</returns>
        public static double getDouble(SqlDataReader R, String Column)
        {
            try
            {
```

```
                object col = getColumn(R, Column);
                if (col != null && col.GetType() != typeof(DBNull))
                {
                    return Convert.ToDouble(col.ToString());
                }
            }
            catch
            {
                return 0.0;
            }
            return 0.0;
        }

        public void CloseConnection()
        {
            _Connection.Close();
        }



        #region IDisposable Members

        public void Dispose()
        {
            _Command.Dispose();
            _Connection.Dispose();
        }

        #endregion
    }
}
```

1.4.3.2 Knowledgebases\Sql\SqlResult.cs

Represents a common Fortress Sql Result. Has an ErrorNumber, ErrorCode (S - Success, H - Error) and an ErrorDescription.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;

namespace FaultDetection.Knowledgebases.Sql
{
    /// <summary>
    /// Represents a common Fortress Sql Result. Has an ErrorNumber, ErrorCode (S -
Success, H - Error) and an ErrorDescription.
    /// </summary>
    public class SqlResult
    {
        private int _errorNumber;
        private String _errorCode, _errorDescription;


        public int ErrorNumber
        {
            get { return this._errorNumber; }
            set { this._errorNumber = value; }
        }

        public String ErrorCode {
```

```
            get { return this._errorCode; }
            set { this._errorCode = value; }
        }

        public String ErrorDescription
        {
            get { return this._errorDescription; }
            set { this._errorDescription = value; }
        }
        /// <summary>
        /// Creates SqlResult object with ErrorCode H (failed), ErrorNumber -1.
        /// </summary>
        public SqlResult()
        {
            this.ErrorCode = "H";
            this.ErrorNumber = -1;
            this.ErrorDescription = "";
        }
        /// <summary>
        /// Creates a SqlResult object based on the error number and description given
        /// </summary>
        /// <param name="errorNumber">the error number</param>
        /// <param name="desc">the error description</param>
        public SqlResult(int errorNumber, string desc)
        {
            this.ErrorCode = (errorNumber == 0) ? "S" : "H";
            this.ErrorNumber = errorNumber;
            this.ErrorDescription = desc;
        }
        /// <summary>
        /// Creates a SqlResult object based on the SqlDataReader given
        /// </summary>
        /// <param name="R">SqlDataReader to load the result from</param>
        public SqlResult(SqlDataReader R)
        {
            loadFromRecord(R);
        }
        /// <summary>
        /// Loads a SqlResult object from the SqlDataReader given
        /// </summary>
        /// <param name="R">SqlDataReader to load the result from</param>
        public void loadFromRecord(SqlDataReader R)
        {
            this.ErrorNumber = SqlExecutor.getInt(R, "ErrorNumber");
            this.ErrorCode = SqlExecutor.getString(R, "ErrorCode");
            this.ErrorDescription = SqlExecutor.getString(R, "ErrorDescription");

            if (String.IsNullOrEmpty(this.ErrorDescription) &&
SqlExecutor.hasColumn(R,"ErrorDesc")) {
                this.ErrorDescription = SqlExecutor.getString(R, "ErrorDesc");
            }
        }
        /// <summary>
        /// Checks whether or not this result is a Success of Failure.
        /// </summary>
        /// <returns>true if the ErrorCode is S (Success), false otherwise.</returns>
        public bool isSuccess()
        {
            return (this.ErrorCode.Equals("S"));
        }
    }
}
```

1.4.4 Modules\FaultDetectionModuleEngineTester.cs

IFaultDetectionModule - Duplicate module to TMCX designed to test the engine (Type ID: 1, Type Name: EngineTester)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;
using System.Text.RegularExpressions;

namespace FaultDetection.Modules
{
    /// <summary>
    /// Duplicate module to TMCX designed to test the engine.
    /// <para>
    /// <list type="bullet">
    /// <item><description>Type ID: 1</description></item>
    /// <item><description>Type Name: EngineTester</description></item>
    /// </list>
    /// </para>
    /// </summary>
    public class FaultDetectionModuleEngineTester : FaultDetectionModuleTMCX,
IFaultDetectionModule
    {
        private const int _typeID = 1;
        private const string _typeName = "EngineTester";

        public new int TypeID
        {
            get { return FaultDetectionModuleEngineTester._typeID; }
        }

        public new string TypeName
        {
            get { return FaultDetectionModuleEngineTester._typeName; }
        }
    }
}
```

1.4.5 Modules\FaultDetectionModuleTMCX.cs

IFaultDetectionModule - Represents a Fortress TMC (or TMCX) Module (Type ID: 2, Type Name: TMCX). Status/State string to use should be a JSON string representing the current state of the TMC.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;

namespace FaultDetection.Modules
{
    /// <summary>
    /// Represents a Fortress TMC (or TMCX) Module
    /// <para>
    /// <list type="bullet">
    /// <item><description>Type ID: 2</description></item>
```

```
    /// <item><description>Type Name: TMCX</description></item>
    /// </list>
    /// </para>
    /// <para>
    /// Status/State string to use should be a JSON string representing the current
state of the TMC.
    /// </para>
    /// </summary>
    public class FaultDetectionModuleTMCX : IFaultDetectionModule
    {
        private const int _typeID = 2;
        private const string _typeName = "TMCX";
        private string _uniqueId, _uniqueName;
        private Dictionary<string, string> _poperties = new Dictionary<string,
string>(1);

        public Dictionary<String, FaultDetectionParam> Attributes = new
Dictionary<String, FaultDetectionParam>(1);
        public Dictionary<String, FaultDetectionParam> Settings = new
Dictionary<String, FaultDetectionParam>(1);
        public string UniqueId
        {
            get { return this._uniqueId; }
            set { this._uniqueId = value; }
        }
        public string UniqueName
        {
            get { return this._uniqueName; }
            set { this._uniqueName = value; }
        }
        public Dictionary<string, string> Properties
        {
            get { return this._poperties; }
            set { this._poperties = value; }
        }
        public int TypeID
        {
            get { return _typeID; }
        }
        public string TypeName
        {
            get { return _typeName; }
        }

        /// <summary>
        /// Loads the current TMC status based on a JSON state string.
        /// </summary>
        /// <param name="status">a JSON string representing the TMCs state</param>
        /// <returns>the number of attributes loaded from the JSON string</returns>
        /// <exception cref="FormatException">thrown when unable to deserialize the
JSON string given</exception>
        public int loadStatus(string status)
        {
            status = tempTMCStatusFix(status);
            status = status.Replace("\\", "\\\\");
            JObject parsed;
            try
            {
                parsed = (JObject)JsonConvert.DeserializeObject(status);

            }
            catch(Exception ex) {
```

```
                throw new FormatException("Status string given does not match
specification (i.e. JSON) ["+ex.Message+"]");
            }
            if (parsed != null)
            {
                foreach (KeyValuePair<string, JToken> kvp in parsed)
                {
                    parseKey(kvp.Key, kvp.Value);
                }
                initSettingsFromStatus();
                return this.Attributes.Count;
            }
            return -1;
        }

        /// <summary>
        /// Gets one of the Status Attributes from the TMC loaded attributes, based on
the attributes name
        /// </summary>
        /// <param name="attributeName">the name of the attribute to get</param>
        /// <returns>FaultDetectionParam representing the attribute (would be of
FaultDetectionParamType NULL if not such attribute exists)</returns>
        public FaultDetectionParam getStatusAttribute(String attributeName)
        {
            foreach (KeyValuePair<String, FaultDetectionParam> attr in Attributes)
            {
                if (attr.Key.Equals(attributeName))
                {
                    return new FaultDetectionParam(attr.Value.ComputedValue,true);
                }
            }
            return new
FaultDetectionParam("",FaultDetection.FaultDetectionParamType.NULL);
        }

        /// <summary>
        /// Finds a TMC State Attribute based using a Regular Expressions pattern
        /// </summary>
        /// <param name="pattern">a regular expression pattern to search for</param>
        /// <returns>FaultDetectionParam represeting the found attribute value (will
return FaultDetectionParamType.NULL if none found)</returns>
        public FaultDetectionParam findStatusAttribute(String pattern)
        {
            foreach (KeyValuePair<String, FaultDetectionParam> attr in Attributes)
            {
                if (Regex.IsMatch(attr.Key,pattern))
                {
                    return new FaultDetectionParam(attr.Value.ComputedValue,true);
                }
            }
            return new FaultDetectionParam("",
FaultDetection.FaultDetectionParamType.NULL);
        }
        /// <summary>
        /// Gets one of the TMCs settings basd on the settings name
        /// </summary>
        /// <param name="settingName">the name of the setting to get</param>
        /// <returns>FaultDetectionParam object representing the given setting
name</returns>
        public FaultDetectionParam getSetting(String settingName)
        {
            foreach (KeyValuePair<String, FaultDetectionParam> setting in Settings)
```

```
            {
                if (setting.Key.Equals(settingName))
                {
                    return new FaultDetectionParam(setting.Value.ComputedValue,true);
                }
            }
            return new FaultDetectionParam("",
FaultDetection.FaultDetectionParamType.NULL);
        }

        /// <summary>
        /// Loads key TMC settins from its current status (i.e. IP/MAC addresses,
etc.)
        /// </summary>
        private void initSettingsFromStatus()
        {
            try
            {
                this.Properties.Add("IP",
this.findStatusAttribute(@"^General\.System\.TCPIP\..*IP$").ComputedValue);
                this.Properties.Add("MAC",
this.findStatusAttribute(@"^General\.System\.TCPIP\..*_MAC_Address$").ComputedValue.Re
place(":", "-"));
                this.UniqueId =
this.findStatusAttribute(@"^General\.System\.TCPIP\..*_MAC_Address$").ComputedValue.Re
place(":","-");
                this.UniqueName =
this.getStatusAttribute("Comm.Comm_Data.Internal_Params.TillName").ComputedValue;
            }
            catch
            {
            }
        }
        /// <summary>
        /// Parses a key from the JSON State string and adds it to the Attributes List
        /// Replaces white spaces " " with "_"
        /// </summary>
        /// <param name="keyName">the key name from the JSON state string</param>
        /// <param name="token">the JToken representing the given key</param>
        private void parseKey(string keyName, JToken token)
        {
            try
            {
                if (token.HasValues)
                {
                    JObject tkObject = token.ToObject<JObject>();
                    foreach (KeyValuePair<string, JToken> kvp in tkObject)
                    {
                        parseKey(keyName + "." + kvp.Key, kvp.Value);
                    }
                }
                else
                {
                    this.Attributes.Add(keyName.Replace(" ", "_"), new
FaultDetectionParam(token.ToString()));
                }
            }
            catch //(Exception ex)
            {
            }
        }
        /// <summary>
```

```
        /// Some older versions of the TMC have a bug reporting the CKKeyString and
3DESKeyString values.
        /// This is a temporary fix for these.
        /// </summary>
        /// <param name="tmcStatus">the old version TMC state string</param>
        /// <returns>a fixed state string with dummy attributes added</returns>
        private static string tempTMCStatusFix(string tmcStatus)
        {
            string ptrn = @"^.*""CKKeyString"" : ""(.*?)             ""Readers
Params"".*$";
            string sptrn = @"""CKKeyString"" : ""(.*?) ";
            string ntmcStatus = tmcStatus;
            Regex regex = new Regex(ptrn, RegexOptions.Singleline |
RegexOptions.IgnoreCase);
            Match match = regex.Match(tmcStatus);
            if (match.Success)
            {
                ntmcStatus = Regex.Replace(tmcStatus, sptrn,
(MatchEvaluator)delegate(System.Text.RegularExpressions.Match m)
                {
                    if (m.Groups.Count >= 1)
                    {
                        string memberName = @"""CKKeyString"" : """ +
m.Groups[1].Value;
                        memberName += "\",\r\n\"3DESKeyString\" : \"\",\r\n";
                        return memberName;
                    }

                    return m.Value;
                });
            }
            ptrn = @"^.*""Command Line Argument/s"" : ""(.*)"",.*";
            sptrn = @"""Command Line Argument/s"" : ""(.*)"",";
            regex = new Regex(ptrn, RegexOptions.Singleline |
RegexOptions.IgnoreCase);
            match = regex.Match(ntmcStatus);
            if (match.Success)
            {
                ntmcStatus = Regex.Replace(ntmcStatus, sptrn,
(MatchEvaluator)delegate(System.Text.RegularExpressions.Match m)
                {
                    if (m.Groups.Count >= 1)
                    {
                        return @"""Command Line Argument/s"" : """",";
                    }

                    return m.Value;
                });
            }
            return ntmcStatus;
        }


    }
}
```

1.4.6 Notifiers\FaultDetectionEmailNotifier.cs

IFaultDetectionNotifier - Represents an Email Sendeing Notifier

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Mail;

namespace FaultDetection.Notifiers
{
    /// <summary>
    /// Represents an Email Sendeing Notifier
    /// </summary>
    public class FaultDetectionEmailNotifier : IFaultDetectionNotifier
    {
        private FaultDetectionCaseCollection _caseCollection = null;
        public List<FaultDetectionTeam> Teams = new List<FaultDetectionTeam>(1);
        private string smtpServer, smtpUsername, smtpPassword, fromName, emailSubject;

        public FaultDetectionCaseCollection CaseCollection
        {
            get { return this._caseCollection; }
            set { this._caseCollection = value; }
        }

        /// <summary>
        /// Creates a new empty FaultDetectionNotification
        /// <para>
        /// Parametless consturctor is needed for this class to be serializable.
        /// </para>
        /// </summary>
        public FaultDetectionEmailNotifier()
        {

        }

        /// <summary>
        /// Creates a new FaultDetectionEmailNotifier object and automatically set its
CaseCollection to the give FaultDetectionCaseCollection
        /// </summary>
        /// <param name="fdCollection">FaultDetectionCaseCollection to use for
generating notifications</param>
        public FaultDetectionEmailNotifier(FaultDetectionCaseCollection fdCollection)
        {
            this.CaseCollection = fdCollection;
        }

        /// <summary>
        /// Loads the SMTP Settings to use to send emails, the given dictionary should
have:
        /// <para>
        /// <list type="bullet">
        /// <item><term>smtpServer:</term><description>The SMTP Server address to
use</description></item>
        /// <item><term>smtpUsername:</term><description>The Username to use to
connect to the SMTP Server</description></item>
        /// <item><term>smtpPassword:</term><description>the user's password to
connecto the SMTP Server</description></item>
        /// <item><term>(optional) fromName:</term><description>the name to use in the
"Form" field of the outgoing notification email.</description></item>
        /// <item><term>(optional) emailSubject:</term>the subject to user in the
```

```
"Subject£ field of the outgoing notification email.</description></item>
        /// </list>
        /// </para>
        /// </summary>
        /// <param name="settings">Dictionary representing the STMP settings to
use</param>
        public void loadSettings(Dictionary<string, string> settings)
        {
            try
            {
                smtpServer = settings["smtpServer"];
                smtpPassword = settings["smtpPassword"];
                smtpUsername = settings["smtpUsername"];
                fromName = (settings.ContainsKey("fromName") &&
!String.IsNullOrEmpty(settings["fromName"]) ? settings["fromName"] : "FGB Fault
Detection");
                emailSubject = (settings.ContainsKey("emailSubject") &&
!String.IsNullOrEmpty(settings["emailSubject"]) ? settings["emailSubject"] : "FGB
Fault Detection - Faults Detected!");
            }
            catch
            {
            }
        }

        /// <summary>
        /// Sends notification based on the CaseCollection loaded
        /// </summary>
        /// <exception cref="ArgumentNullException">Thrown if no Case Collection was
ever loaded, or some of the SMTP settings are missing.</exception>
        /// <exception cref="ApplicationException">Thrown if the SMTP server failed
sending the Email</exception>
        /// <exception cref="Exception">Thrown if an underlying exception
occured</exception>
        public void sendNotifications()
        {
            if (this._caseCollection == null)
            {
                throw new ArgumentNullException("No Fault Detection Case Collection
Loaded!");
            }
            if (String.IsNullOrEmpty(smtpServer))
            {
                throw new ArgumentNullException("No SMTP Server given in Settings
Dictionary!");
            }
            if (String.IsNullOrEmpty(smtpUsername))
            {
                throw new ArgumentNullException("No SMTP Username given in Settings
Dictionary!");
            }
            if (String.IsNullOrEmpty(smtpPassword))
            {
                throw new ArgumentNullException("No SMTP Password given in Settings
Dictionary!");
            }
            try
            {

                SmtpClient mySmtpClient = new SmtpClient(smtpServer);
                mySmtpClient.UseDefaultCredentials = false;
                System.Net.NetworkCredential basicAuthenticationInfo = new
```

```
System.Net.NetworkCredential(smtpUsername, smtpPassword);
                mySmtpClient.Credentials = basicAuthenticationInfo;

                MailAddress from = new MailAddress(smtpUsername, fromName);

                foreach (FaultDetectionTeam team in
this._caseCollection.NotificationTeams)
                {
                    if (team.Faults != null && team.Faults.Count > 0)
                    {
                        string emailContent = getTeamEmailContent(team);
                        if (!string.IsNullOrEmpty(emailContent))
                        {
                            MailAddress to = new MailAddress(team.Email, team.Name);
                            MailMessage myMail = new System.Net.Mail.MailMessage(from,
to);
                            myMail.Subject = emailSubject;
                            myMail.SubjectEncoding = System.Text.Encoding.UTF8;
                            myMail.Body = emailContent;
                            myMail.BodyEncoding = System.Text.Encoding.UTF8;
                            myMail.IsBodyHtml = true;
                            mySmtpClient.Send(myMail);
                        }
                    }
                }
            }
            catch (SmtpException ex)
            {
              throw new ApplicationException
                ("SmtpException has occured: " + ex.Message);
            }
            catch (Exception ex)
            {
               throw ex;
            }
        }

        /// <summary>
        /// Generates and returns the email content for the given team
        /// </summary>
        /// <param name="team">the fault detection team to notify</param>
        /// <returns>the content (body) of the notification email to send</returns>
        private string getTeamEmailContent(FaultDetectionTeam team)
        {
            if (this.CaseCollection != null)
            {
                string teamContent = null;
                if (team != null && team.Faults != null && team.Faults.Count > 0)
                {
                    teamContent = generateTeamHtmlContent(team);
                }
                if (!string.IsNullOrEmpty(teamContent))
                {
                    StringBuilder sb = new StringBuilder();
                    sb.AppendLine("<h1>" + _caseCollection.Module.TypeName + " - " +
_caseCollection.Module.UniqueName + " (" + _caseCollection.Module.UniqueId +
")</h1>");
                    sb.AppendLine("<table><tbody>");
                    foreach (KeyValuePair<string, string> prop in
_caseCollection.Module.Properties)
                    {
                        sb.AppendLine("<tr><td>" + prop.Key + "</td><td>" + prop.Value
```

```
+ "</td></tr>");
                    }
                    sb.AppendLine("</tbody></table>");
                    if (!_caseCollection.Result.Passed)
                    {
                        sb.AppendLine("<p>Collection Error: <strong>" +
_caseCollection.Result.Error + "</strong></p>");
                    }


                    sb.AppendLine(teamContent);

                    return sb.ToString();
                }
            }
            return null;
        }

        /// <summary>
        /// Generates/returns the html of the faults found to use in the notification
email.
        /// </summary>
        /// <param name="team">the fault detection team the notification is
for</param>
        /// <returns>html representing all the fault found for the given
team</returns>
        private string generateTeamHtmlContent(FaultDetectionTeam team)
        {
            if (team != null)
            {
                if (team.Faults == null || team.Faults.Count < 1)
                {
                    return null;
                }

                StringBuilder sb = new StringBuilder();
                int faults = 0;
                foreach (FaultDetectionCase fault in team.Faults)
                {
                    if (fault.Result.Executed && !fault.Result.Passed && ((fault.Flags
& FaultDetectionCase.FaultDetectionCaseFlags.NO_NOTIFICATIONS) == 0))
                    {
                        sb.AppendLine("<h2>" + fault.Name + " (" + fault.ID +
")</h2>");
                        if (!fault.Result.Passed)
                        {
                            sb.AppendLine("<h3>Error: " + fault.Result.Error +
"</h3>");
                        }
                        if (!String.IsNullOrEmpty(fault.Description))
                        {
                            sb.AppendLine("<p>" + fault.Description + "</p>");
                        }

sb.AppendLine("<table><thead><tr><td>ID</td><td>Name</td><td>Params</td><td>Error</td>
<td>Message</td></tr></thead><tbody>");

                        foreach (FaultDetectionTest test in fault.Tests)
                        {
                            if (test.Result.Executed && !test.Result.Passed)
                            {
                                sb.AppendLine("<tr>");
```

```
                                sb.AppendLine("<td>" + test.ID + "</td>");
                                sb.AppendLine("<td>" + test.Name + "</td>");
                                sb.AppendLine("<td>(");
                                if (test.Rules != null && test.Rules.Count > 0)
                                {
                                    int lastGroup = 0;
                                    bool lastMerged = true;
                                    FaultDetectionRelationType lastGroupRelation = 0,
lastRelation = 0;
                                    sb.AppendLine("((" +
getParamDetails(test.Rules[0].Param1) + " <div style='padding-left: 20px; font-style:
italic'>" + (test.Rules[0].Negate ? "NOT " : "") + test.Rules[0].OperationTypeName +
"</div>" + getParamDetails(test.Rules[0].Param2) + ")");
                                    lastRelation = test.Rules[0].RelationType;
                                    lastGroupRelation = test.Rules[0].RelationType;
                                    for (int i = 1; i < test.Rules.Count; i++)
                                    {
                                        //Log.i(TAG," Testing params: "+i);
                                        FaultDetectionTestRule parameters =
test.Rules[i];
                                        if (parameters.Group != lastGroup)
                                        {
                                            sb.AppendLine(")<br/><strong> " +
test.Rules[i - 1].RelationType + "</strong><br/> (");
                                        }
                                        sb.Append(((parameters.Group == lastGroup) ?
"<div><strong>" + lastRelation.ToString() + "</strong></div>" : "") + " (" +
getParamDetails(parameters.Param1) + " <div style='padding-left: 20px; font-style:
italic'>" + (parameters.Negate ? "NOT " : "") + parameters.OperationTypeName +
"</div>" + getParamDetails(parameters.Param2) + ")");
                                        if (parameters.Group != lastGroup)
                                        {
                                            lastGroupRelation = test.Rules[i -
1].RelationType;
                                            lastMerged = true;
                                        }
                                        lastRelation = parameters.RelationType;
                                        lastGroup = parameters.Group;
                                    }
                                    if (!lastMerged)
                                    {
                                        //incase the last one is a group as well, we
need to merge it...
                                        sb.Append(")");
                                    }
                                    sb.AppendLine(")<br/>");
                                }

                                sb.AppendLine(")</td>");
                                sb.AppendLine("<td>" + test.Result.Error + "</td>");
                                sb.AppendLine("<td>" + test.FailMessage + "</td>");
                                sb.AppendLine("</tr>");
                            }
                        }
                        sb.AppendLine("</tbody></table>");
                        faults++;
                    }

                }
                if (faults > 0)
                {
                    return sb.ToString();
```

```
                }
            }
            return null;
        }
        /// <summary>
        /// Returns a HTML string representing the given FaultDetectionParam complete
information/details.
        /// </summary>
        /// <param name="param">a FaultDetectionParam to parse</param>
        /// <returns>the given FaultDetectionParam infroamtion/details</returns>
        private String getParamDetails(FaultDetectionParam param)
        {
            return param.Value + " <span style='font-size: small'>[" + param.Type +
(param.Type == FaultDetectionParamType.MODULE_ATTRIBUTE || param.Type ==
FaultDetectionParamType.TEST_ID || param.Type == FaultDetectionParamType.FUNCTION ||
param.Type == FaultDetectionParamType.ENVIRONMENT_ATTRIBUTE ? " = " +
param.ComputedValue + " [" + param.ComputedType + "]" : "") + "]</span>";
        }
    }
}
```

1.4.7 Operations

The below list is all the IFaultDetectionOperation impelemtations currently used in this project. They are all under the "FaultDetection.Operations" namespace and are loaded dynamically using Reflection at runtime (during the initialisation a FaultDetectionEngine instance).

1.4.7.1 FaultDetectionOperationContains.cs

Checks if one FaultDetectionParam (computed value) contains a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- STRING, STRING
- STRING_ARRAY, STRING
- STRING_ARRAY, REGEX_PATTERN
- INT_ARRAY, INTEGER
- DECIMAL_ARRAY, DECIMAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a Containts operation:
    ///
    /// Examaple:
    /// evaluate("XYZ","Y") will pass
    /// evaluate("XYZ","R") will fail
    /// evaluate("Y","XYZ") will faile
    /// </summary>
    class FaultDetectionOperationContains : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) contains a 2nd
FaultDetectionParam (computed value)
        ///
```

```
        /// Valid FaultDetectionParamType combinations:
        ///     STRING, STRING
        ///     STRING_ARRAY, STRING
        ///     STRING_ARRAY, REGEX_PATTERN
        ///     INT_ARRAY, INTEGER
        ///     DECIMAL_ARRAY, DECIMAL
        /// </summary>
        /// <param name="one">the containing param</param>
        /// <param name="two">the param to search for</param>
        /// <returns>FaultDetectionResult.Passed = True if 1st param contains 2nd,
false otherwise</returns>
        public FaultDetectionResult evaluate(FaultDetectionParam one,
FaultDetectionParam two)
        {
            //Log.i(TAG,"Contains test: "+one.getType() +" vs " + two.getType());

            if (one == null || two == null) return new FaultDetectionResult(false,
"both parameters are null");

            switch (one.ComputedType)
            {
                case FaultDetectionParamType.STRING:
                    //Log.i(TAG,one.getString()+ " /contains: "+two.getString());
                    if (two.ComputedType == FaultDetectionParamType.STRING &&
one.get<String>() != null)
                    {
                        return new
FaultDetectionResult(one.get<String>().IndexOf(two.get<String>()) > -1);
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.STRING_ARRAY:
                    //System.out.println("One is String Array");
                    string[] oneStrArray = one.get<String[]>();
                    string twoString = two.get<String>();
                    if (oneStrArray != null && oneStrArray.Length > 0 &&
!String.IsNullOrEmpty(twoString))
                    {
                        switch (two.ComputedType)
                        {
                            case FaultDetectionParamType.STRING:
                                foreach (string si in oneStrArray)
                                {
                                    if (si.Equals(twoString)) return new
FaultDetectionResult(true);
                                }
                                return new FaultDetectionResult(false);
                            case FaultDetectionParamType.REGEX_PATTERN:
                                foreach (string si in oneStrArray)
                                {
                                    //System.out.println("    -> "+si+" /
"+two.getString());
                                    if (Regex.IsMatch(si, twoString)) return new
FaultDetectionResult(true);
                                }
                                return new FaultDetectionResult(false);
                            default:
                                return
FaultDetectionResult.invalidParameterOperator(one, two, this);
                        }

                    }
```

```
                    return new FaultDetectionResult(false, "parameter 1 (" +
one.ComputedValue + ") isnt a valid STRING_ARRAY or parameter 2 (" + twoString + ")
isnt a valid string");
                case FaultDetectionParamType.INT_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.INTEGER)
                    {
                        int[] oneIntArray = one.get<int[]>();
                        if (oneIntArray != null && oneIntArray.Length > 0)
                        {
                            int twoInt = two.get<Int32>();
                            foreach (int x in oneIntArray)
                            {
                                if (x == twoInt) return new
FaultDetectionResult(true);
                            }
                            return new FaultDetectionResult(false);
                        }
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.DECIMAL_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                    {
                        Decimal[] oneDecArray = one.get<Decimal[]>();
                        if (oneDecArray != null && oneDecArray.Length > 0)
                        {
                            Decimal twoDecimal = two.get<Decimal>();
                            foreach (Decimal x in oneDecArray)
                            {
                                if (x == twoDecimal) return new
FaultDetectionResult(true);
                            }
                            return new FaultDetectionResult(false);
                        }
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
            }
            return new FaultDetectionResult(false, "paramter 1 type (" + one.Type + "
== " + one.ComputedType + ") is invalid for CONTAINS operation");
        }
    }
}
```

1.4.7.2 FaultDetectionOperationEquals.cs

Checks if one FaultDetectionParam (computed value) equals a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- NULL, NULL
- STRING, STRING
- STRING, REGEX_PATTERN
- REGEX_PATTERN, STRING
- INTEGER, INTEGER
- INTEGER, DECIMAL
- DECIMAL, DECIMAL
- DECIMAL, INTEGER
- BOOLEAN, BOOLEAN
- INT_ARRAY,INT_ARRAY
- DEICMAL_ARRAY, DECIMAL_ARRAY
- STRING_ARRAY,STRING_ARRAY
- DATE,DATE
- TIMESPAN, TIMESPAN

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents an Equality operation:
    ///
    /// Examaple:
    /// evaluate("Y","Y") will pass
    /// evaluate("Y","XYZ") will fail
    /// evaluate(1,1) will pass
    /// evaluate(1,2) will fail
    /// evaluate([1,2,3],[1,2,3]) will pass
    /// evaluate([1,2,3],[1,2,4]) will fail
    /// </summary>
    class FaultDetectionOperationEquals : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) equals a 2nd
FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
        ///     NULL, NULL
        ///     STRING, STRING
        ///     STRING, REGEX_PATTERN
        ///     REGEX_PATTERN, STRING
        ///     INTEGER, INTEGER
        ///     INTEGER, DECIMAL
        ///     DECIMAL, DECIMAL
        ///     DECIMAL, INTEGER
        ///     BOOLEAN, BOOLEAN
        ///     INT_ARRAY,INT_ARRAY
```

```
            ///     DEICMAL_ARRAY, DECIMAL_ARRAY
            ///     STRING_ARRAY,STRING_ARRAY
            ///     DATE,DATE
            ///     TIMESPAN, TIMESPAN
            /// </summary>
            /// <param name="one">a base FaultDetectionParam to compare</param>
            /// <param name="two">a FaultDetectionParam to compare to</param>
            /// <returns>FaultDetectionResult.Passed = True if 1st param
    equals/matches 2nd, false otherwise</returns>
            public FaultDetectionResult evaluate(FaultDetectionParam one,
    FaultDetectionParam two)
            {

                switch (one.ComputedType)
                {
                    case FaultDetectionParamType.NULL:
                        if (two.ComputedType == FaultDetectionParamType.NULL)
                        {
                            return new FaultDetectionResult(true);
                        }
                        return FaultDetectionResult.invalidParameterOperator(one, two,
    this);
                    case FaultDetectionParamType.STRING:
                        if (two.ComputedType == FaultDetectionParamType.STRING)
                        {
                            return new
    FaultDetectionResult(one.get<String>().Equals(two.get<String>()));
                        }
                        else if (two.ComputedType ==
    FaultDetectionParamType.REGEX_PATTERN)
                        {
                            return new
    FaultDetectionResult(Regex.IsMatch(one.get<String>(), two.get<String>()));
                        }
                        return FaultDetectionResult.invalidParameterOperator(one, two,
    this);
                    case FaultDetectionParamType.INTEGER:
                        if (two.ComputedType == FaultDetectionParamType.INTEGER)
                        {
                            return new FaultDetectionResult(one.get<Int32>() ==
    two.get<Int32>());
                        }
                        else if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                        {
                            return new FaultDetectionResult(one.get<Int32>() ==
    two.get<Decimal>());
                        }
                        return FaultDetectionResult.invalidParameterOperator(one, two,
    this);
                    case FaultDetectionParamType.DECIMAL:
                        if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                        {
                            return new FaultDetectionResult(one.get<Decimal>() ==
    two.get<Decimal>());
                        }
                        else if (two.ComputedType == FaultDetectionParamType.INTEGER)
                        {
                            return new FaultDetectionResult(one.get<Decimal>() ==
    two.get<Int32>());
                        }
                        return FaultDetectionResult.invalidParameterOperator(one, two,
    this);
```

```
                    case FaultDetectionParamType.BOOLEAN:
                        if (two.ComputedType == FaultDetectionParamType.BOOLEAN)
                        {
                            return new FaultDetectionResult(one.get<Boolean>() ==
two.get<Boolean>());
                        }
                        return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                    case FaultDetectionParamType.STRING_ARRAY:
                        if (two.ComputedType == FaultDetectionParamType.STRING_ARRAY)
                        {
                            String[] oneStrArray = one.get<String[]>();
                            if (oneStrArray != null && oneStrArray.Length > 0)
                            {
                                String[] twoStrArray = two.get<String[]>();
                                if (twoStrArray != null && twoStrArray.Length ==
oneStrArray.Length)
                                {
                                    for (int i = 0; i < twoStrArray.Length; i++)
                                    {
                                        if (!oneStrArray[i].Equals(twoStrArray[i]))
return new FaultDetectionResult(false, "array items at index " + i + " (" +
oneStrArray[i] + " / " + twoStrArray[i] + ") dont match");
                                    }
                                    return new FaultDetectionResult(true);
                                }
                            }
                        }
                        return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                    case FaultDetectionParamType.INT_ARRAY:
                        if (two.ComputedType == FaultDetectionParamType.INT_ARRAY)
                        {
                            int[] oneIntArray = one.get<int[]>();
                            if (oneIntArray != null && oneIntArray.Length > 0)
                            {
                                int[] twoIntArray = two.get<int[]>();
                                if (twoIntArray != null && twoIntArray.Length ==
oneIntArray.Length)
                                {
                                    for (int i = 0; i < twoIntArray.Length; i++)
                                    {
                                        if (oneIntArray[i] != twoIntArray[i]) return
new FaultDetectionResult(false, "array items at index " + i + " (" +
oneIntArray[i] + " / " + twoIntArray[i] + ") dont match");
                                    }
                                    return new FaultDetectionResult(true);
                                }
                            }
                        }
                        return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                    case FaultDetectionParamType.DECIMAL_ARRAY:
                        if (two.ComputedType == FaultDetectionParamType.DECIMAL_ARRAY)
                        {
                            Decimal[] oneArray = one.get<Decimal[]>();
                            if (oneArray != null && oneArray.Length > 0)
                            {
                                Decimal[] twoArray = two.get<Decimal[]>();
                                if (twoArray != null && twoArray.Length ==
oneArray.Length)
                                {
```

```
                                for (int i = 0; i < twoArray.Length; i++)
                                {
                                    if (oneArray[i] != twoArray[i]) return new
FaultDetectionResult(false, "array items at index " + i + " (" + oneArray[i] + " /
" + twoArray[i] + ") dont match");
                                }
                                return new FaultDetectionResult(true);
                            }
                        }
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.REGEX_PATTERN:
                    if (two.ComputedType == FaultDetectionParamType.STRING)
                    {
                        return new
FaultDetectionResult(Regex.IsMatch(two.get<String>(), one.get<String>()));
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.DATE:
                    if (two.ComputedType == FaultDetectionParamType.DATE)
                    {
                        return new FaultDetectionResult(one.get<DateTime>() ==
two.get<DateTime>());
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.TIMESPAN:
                    if (two.ComputedType == FaultDetectionParamType.TIMESPAN)
                    {
                        return new FaultDetectionResult(one.get<TimeSpan>() ==
two.get<TimeSpan>());
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);

            }
            return new FaultDetectionResult(false, "Paramter 1 (" + one.Type + "
== " + one.ComputedType + ") invalid for equality operation");
        }
    }
}
```

•

1.4.7.3 FaultDetectionOperationGreaterEquals.cs

Checks if one FaultDetectionParam (computed value) is greater than or equales (>=) to a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- INTEGER, INTEGER
- INTEGER, DECIMAL
- DECIMAL, DECIMAL
- DECIMAL, INTEGER
- DATE, DATE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a "Greater or Equals To" (>=) operation:
    ///
    /// Examaple:
    /// evaluate(1,1) will pass
    /// evaluate(2,1) will pass
    /// evaluate('2014-01-01','2014-01-01') will pass
    /// evaluate(1,2) will fail
    /// </summary>
    class FaultDetectionOperationGreaterEquals : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) is greater than or
equales (>=) to a 2nd FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
        ///     INTEGER, INTEGER
        ///     INTEGER, DECIMAL
        ///     DECIMAL, DECIMAL
        ///     DECIMAL, INTEGER
        ///     DATE, DATE
        /// </summary>
        /// <param name="one">a base FaultDetectionParam to compare</param>
        /// <param name="two">a FaultDetectionParam to compare to</param>
        /// <returns>FaultDetectionResult.Passed = True if 1st param greater
than/equals (>=) 2nd, false otherwise</returns>
        public FaultDetectionResult evaluate(FaultDetectionParam one,
FaultDetectionParam two)
        {
            if (one.ComputedType == FaultDetectionParamType.INTEGER)
            {
                if (two.ComputedType == FaultDetectionParamType.INTEGER)
                {
                    return new FaultDetectionResult(one.get<Int32>() >=
two.get<Int32>());
                }
                else if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                {
                    return new FaultDetectionResult(one.get<Int32>() >=
two.get<Decimal>());
                }
```

```
            }
            else if (one.ComputedType == FaultDetectionParamType.DECIMAL)
            {
                if (two.ComputedType == FaultDetectionParamType.DECIMAL) {
                    return new FaultDetectionResult((one.get<Decimal>() >=
two.get<Decimal>()));
                }
                else if (two.ComputedType == FaultDetectionParamType.INTEGER)
                {
                    return new FaultDetectionResult((one.get<Decimal>() >=
two.get<Int32>()));
                }
            }
            else if (one.ComputedType == FaultDetectionParamType.DATE &&
two.ComputedType == FaultDetectionParamType.DATE)
            {
                return new FaultDetectionResult((one.get<DateTime>() >=
two.get<DateTime>()));
            }
            return FaultDetectionResult.invalidParameterOperator(one, two, this);
        }

    }
}
```

1.4.7.4 FaultDetectionOperationGreaterThan.cs

Checks if one FaultDetectionParam (computed value) is greater than (>) to a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- INTEGER, INTEGER
- INTEGER, DECIMAL
- DECIMAL, DECIMAL
- DECIMAL, INTEGER
- DATE, DATE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a "Greater Than" (>) operation:
    ///
    /// Examaple:
    /// evaluate(2,1) will pass
    /// evaluate('2014-01-02','2014-01-01') will pass
    /// evaluate(1,1) will fail
    /// evaluate(1,2) will fail
    /// </summary>
    class FaultDetectionOperationGreaterThan : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) is greater than (>)
to a 2nd FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
        ///     INTEGER, INTEGER
```

```
            ///     INTEGER, DECIMAL
            ///     DECIMAL, DECIMAL
            ///     DECIMAL, INTEGER
            ///     DATE, DATE
            /// </summary>
            /// <param name="one">a base FaultDetectionParam to compare</param>
            /// <param name="two">a FaultDetectionParam to compare to</param>
            /// <returns>FaultDetectionResult.Passed = True if 1st param greater than
    (>) 2nd, false otherwise</returns>
            public FaultDetectionResult evaluate(FaultDetectionParam one,
    FaultDetectionParam two)
            {
                if (one.ComputedType == FaultDetectionParamType.INTEGER)
                {
                    if (two.ComputedType == FaultDetectionParamType.INTEGER) {
                        return new FaultDetectionResult(one.get<Int32>() >
    two.get<Int32>());
                    }
                    else if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                    {
                        return new FaultDetectionResult(one.get<Int32>() >
    two.get<Decimal>());
                    }
                }
                else if (one.ComputedType == FaultDetectionParamType.DECIMAL)
                {
                    if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                    {
                        return new FaultDetectionResult((one.get<Decimal>() >
    two.get<Decimal>()));
                    }
                    else if (two.ComputedType == FaultDetectionParamType.INTEGER)
                    {
                        return new FaultDetectionResult((one.get<Decimal>() >
    two.get<Int32>()));
                    }
                }
                else if (one.ComputedType == FaultDetectionParamType.DATE &&
    two.ComputedType == FaultDetectionParamType.DATE)
                {
                    return new FaultDetectionResult((one.get<DateTime>() >
    two.get<DateTime>()));
                }
                return FaultDetectionResult.invalidParameterOperator(one, two, this);
            }

        }
    }

    •
```

1.4.7.5 FaultDetectionOperationLessEquals.cs

Checks if one FaultDetectionParam (computed value) is less than or equales to a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- INTEGER, INTEGER
- INTEGER, DECIMAL
- DECIMAL, DECIMAL
- DECIMAL, INTEGER
- DATE, DATE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a "Less or Equals To" operation:
    ///
    /// Examaple:
    /// evaluate(1,1) will pass
    /// evaluate(1,2) will pass
    /// evaulate('2014-01-01','2014-01-02') will pass
    /// evaluate(2,1) will fail
    /// </summary>
    class FaultDetectionOperationLessEquals : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) is less than or
equales to a 2nd FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
        ///     INTEGER, INTEGER
        ///     INTEGER, DECIMAL
        ///     DECIMAL, DECIMAL
        ///     DECIMAL, INTEGER
        ///     DATE, DATE
        /// </summary>
        /// <param name="one">a base FaultDetectionParam to compare</param>
        /// <param name="two">a FaultDetectionParam to compare to</param>
        /// <returns>FaultDetectionResult.Passed = True if 1st param less than or
equals to a 2nd param, false otherwise</returns>
        public FaultDetectionResult evaluate(FaultDetectionParam one,
FaultDetectionParam two)
        {
            if (one.ComputedType == FaultDetectionParamType.INTEGER)
            {
                if (two.ComputedType == FaultDetectionParamType.INTEGER) {
                    return new FaultDetectionResult(one.get<Int32>() <=
two.get<Int32>());
                }
                else if (two.ComputedType == FaultDetectionParamType.DECIMAL) {
                    return new FaultDetectionResult(one.get<Int32>() <=
two.get<Decimal>());
                }
            }
            else if (one.ComputedType == FaultDetectionParamType.DECIMAL)
            {
```

```
                if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                {
                    return new FaultDetectionResult((one.get<Decimal>() <=
two.get<Decimal>()));
                }
                else if (two.ComputedType == FaultDetectionParamType.INTEGER)
                {
                    return new FaultDetectionResult((one.get<Decimal>() <=
two.get<Int32>()));
                }
            }
            else if (one.ComputedType == FaultDetectionParamType.DATE &&
two.ComputedType == FaultDetectionParamType.DATE)
            {
                return new FaultDetectionResult((one.get<DateTime>() <=
two.get<DateTime>()));
            }
            return FaultDetectionResult.invalidParameterOperator(one, two, this);
        }

    }
}

•
```

1.4.7.6 FaultDetectionOperationLessThan.cs

Checks if one FaultDetectionParam (computed value) is less than to a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- INTEGER, INTEGER
- INTEGER, DECIMAL
- DECIMAL, DECIMAL
- DECIMAL, INTEGER
- DATE, DATE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a "Less Than" operation:
    ///
    /// Examaple:
    /// evaluate(1,1) will pass
    /// evaluate(1,2) will pass
    /// evaluate('2014-01-01','2014-01-02') will pass
    /// evaluate(2,1) will fail
    /// </summary>
    class FaultDetectionOperationLessThan : IFaultDetectionOperation
    {

        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) is less than to a
2nd FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
```

```
            ///     INTEGER, INTEGER
            ///     INTEGER, DECIMAL
            ///     DECIMAL, DECIMAL
            ///     DECIMAL, INTEGER
            ///     DATE, DATE
            /// </summary>
            /// <param name="one">a base FaultDetectionParam to compare</param>
            /// <param name="two">a FaultDetectionParam to compare to</param>
            /// <returns>FaultDetectionResult.Passed = True if 1st param less than to
    a 2nd param, false otherwise</returns>
            public FaultDetectionResult evaluate(FaultDetectionParam one,
    FaultDetectionParam two)
            {
                // using if/else if since both param types need to be the same
                if (one.ComputedType == FaultDetectionParamType.INTEGER)
                {
                    if (two.ComputedType == FaultDetectionParamType.INTEGER)
                    {
                        return new FaultDetectionResult((one.get<Int32>() <
    two.get<Int32>()));
                    }
                    else if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                    {
                        return new FaultDetectionResult((one.get<Int32>() <
    two.get<Decimal>()));
                    }
                }
                else if (one.ComputedType == FaultDetectionParamType.DECIMAL)
                {
                    if (two.ComputedType == FaultDetectionParamType.DECIMAL)
                    {
                        return new FaultDetectionResult((one.get<Decimal>() <
    two.get<Decimal>()));
                    }
                    else if (two.ComputedType == FaultDetectionParamType.INTEGER)
                    {
                        return new FaultDetectionResult((one.get<Decimal>() <
    two.get<Int32>()));
                    }
                }
                else if (one.ComputedType == FaultDetectionParamType.DATE &&
    two.ComputedType == FaultDetectionParamType.DATE)
                {
                    return new FaultDetectionResult((one.get<DateTime>() <
    two.get<DateTime>()));
                }
                return FaultDetectionResult.invalidParameterOperator(one, two, this);
            }

        }
    }
```

•

1.4.7.7 FaultDetectionOperationLongerEquals.cs

Checks if one FaultDetectionParam (computed value) is longer than or equals (length) to a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- STRING, STRING
- STRING_ARRAY, STRING_ARRAY
- INT_ARRAY, INT_ARRAY
- DECIMAL_ARRAY, DECIMAL_ARRAY
- TIMESPAN, TIMESPAN

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a "Longer Than or Equal (Length)" operation:
    ///
    /// Examaple:
    /// evaluate("XZ","Z") will pass
    /// evaluate([1,2,3],[1,2]) will pass
    /// evaluate("R","XX") will fail
    /// </summary>
    class FaultDetectionOperationLongerEquals : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) is longer than or
equals (length) to a 2nd FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
        ///     STRING, STRING
        ///     STRING_ARRAY, STRING_ARRAY
        ///     INT_ARRAY, INT_ARRAY
        ///     DECIMAL_ARRAY, DECIMAL_ARRAY
        ///     TIMESPAN, TIMESPAN
        /// </summary>
        /// <param name="one">a base FaultDetectionParam to compare</param>
        /// <param name="two">a FaultDetectionParam to compare to</param>
        /// <returns>FaultDetectionResult.Passed = True if 1st param longer than or
equals (length) to a 2nd param, false otherwise</returns>
        public FaultDetectionResult evaluate(FaultDetectionParam one,
FaultDetectionParam two)
        {
            switch (one.ComputedType)
            {
                case FaultDetectionParamType.STRING:
                    if (two.ComputedType == FaultDetectionParamType.STRING)
                    {
                        return new FaultDetectionResult(one.get<String>().Length >=
two.get<String>().Length);
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.STRING_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.STRING_ARRAY)
                    {
                        string[] oneStrArray = one.get<String[]>();
```

```
                        string[] twoStrArray = two.get<String[]>();
                        if (oneStrArray != null && twoStrArray != null)
                        {
                            return new FaultDetectionResult(oneStrArray.Length >=
twoStrArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid STRING_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.INT_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.INT_ARRAY)
                    {
                        int[] oneIntArray = one.get<int[]>();
                        int[] twoIntArray = two.get<int[]>();
                        if (oneIntArray != null && twoIntArray != null)
                        {
                            return new FaultDetectionResult(oneIntArray.Length >=
twoIntArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid INT_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.DECIMAL_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.DECIMAL_ARRAY)
                    {
                        Decimal[] oneArray = one.get<Decimal[]>();
                        Decimal[] twoArray = two.get<Decimal[]>();
                        if (oneArray != null && twoArray != null)
                        {
                            return new FaultDetectionResult(oneArray.Length >=
twoArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid DECIMAL_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.TIMESPAN:
                    if (two.ComputedType == FaultDetectionParamType.TIMESPAN)
                    {
                        return new FaultDetectionResult(one.get<TimeSpan>() >=
two.get<TimeSpan>());
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
            }
            return new FaultDetectionResult(false, "paramter 1 type (" + one.Type + "
== " + one.ComputedType + ") is invalid for LONGER_EQUALS operation");
        }

    }
}
```

1.4.7.8 FaultDetectionOperationLongerThan.cs

Checks if one FaultDetectionParam (computed value) is longer than (length) to a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- STRING, STRING
- STRING_ARRAY, STRING_ARRAY
- INT_ARRAY, INT_ARRAY
- DECIMAL_ARRAY, DECIMAL_ARRAY
- TIMESPAN, TIMESPAN

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a "Longer Than" operation:
    ///
    /// Examaple:
    /// evaluate("XZ","Z") will pass
    /// evaluate([1,2,3],[1,2]) will pass
    /// evaluate("R","X") will fail
    /// </summary>
    class FaultDetectionOperationLongerThan : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) is longer than (length)
to a 2nd FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
        ///     STRING, STRING
        ///     STRING_ARRAY, STRING_ARRAY
        ///     INT_ARRAY, INT_ARRAY
        ///     DECIMAL_ARRAY, DECIMAL_ARRAY
        ///     TIMESPAN, TIMESPAN
        /// </summary>
        /// <param name="one">a base FaultDetectionParam to compare</param>
        /// <param name="two">a FaultDetectionParam to compare to</param>
        /// <returns>FaultDetectionResult.Passed = True if 1st param longer than
(length) to a 2nd param, false otherwise</returns>
        public FaultDetectionResult evaluate(FaultDetectionParam one,
FaultDetectionParam two)
        {
            switch (one.ComputedType)
            {
                case FaultDetectionParamType.STRING:
                    if (two.ComputedType == FaultDetectionParamType.STRING)
                    {
                        return new FaultDetectionResult(one.get<String>().Length >
two.get<String>().Length);
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.STRING_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.STRING_ARRAY)
                    {
                        string[] oneStrArray = one.get<String[]>();
```

```
                        string[] twoStrArray = two.get<String[]>();
                        if (oneStrArray != null && twoStrArray != null)
                        {
                            return new FaultDetectionResult(oneStrArray.Length >
twoStrArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid STRING_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.INT_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.INT_ARRAY)
                    {
                        int[] oneIntArray = one.get<int[]>();
                        int[] twoIntArray = two.get<int[]>();
                        if (oneIntArray != null && twoIntArray != null)
                        {
                            return new FaultDetectionResult(oneIntArray.Length >
twoIntArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid INT_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.DECIMAL_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.DECIMAL_ARRAY)
                    {
                        Decimal[] oneArray = one.get<Decimal[]>();
                        Decimal[] twoArray = two.get<Decimal[]>();
                        if (oneArray != null && twoArray != null)
                        {
                            return new FaultDetectionResult(oneArray.Length >
twoArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid DECIMAL_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.TIMESPAN:
                    if (two.ComputedType == FaultDetectionParamType.TIMESPAN)
                    {
                        return new FaultDetectionResult(one.get<TimeSpan>() >
two.get<TimeSpan>());
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
            }
            return new FaultDetectionResult(false, "paramter 1 type (" + one.Type + "
== " + one.ComputedType + ") is invalid for LONGER_THAN operation");
        }

    }
}
```

1.4.7.9 FaultDetectionOperationShorterEquals.cs

Checks if one FaultDetectionParam (computed value) is shorter than or equals (length) to a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- STRING, STRING
- STRING_ARRAY, STRING_ARRAY
- INT_ARRAY, INT_ARRAY
- DECIMAL_ARRAY, DECIMAL_ARRAY
- TIMESPAN, TIMESPAN

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a "Shorter Than or Equals (length)" operation:
    ///
    /// Examaple:
    /// evaluate("XZ","XXZ") will pass
    /// evaluate([1,2,3],[1,2,3,4]) will pass
    /// evaluate("RX","X") will fail
    /// </summary>
    class FaultDetectionOperationShorterEquals : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) is shorter than or
equals (length) to a 2nd FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
        ///     STRING, STRING
        ///     STRING_ARRAY, STRING_ARRAY
        ///     INT_ARRAY, INT_ARRAY
        ///     DECIMAL_ARRAY, DECIMAL_ARRAY
        ///     TIMESPAN, TIMESPAN
        /// </summary>
        /// <param name="one">a base FaultDetectionParam to compare</param>
        /// <param name="two">a FaultDetectionParam to compare to</param>
        /// <returns>FaultDetectionResult.Passed = True if 1st param shorter than or
equals (length) to a 2nd param, false otherwise</returns>
        public FaultDetectionResult evaluate(FaultDetectionParam one,
FaultDetectionParam two)
        {
            switch (one.ComputedType)
            {
                case FaultDetectionParamType.STRING:
                    if (two.ComputedType == FaultDetectionParamType.STRING)
                    {
                        return new FaultDetectionResult(one.get<String>().Length <=
two.get<String>().Length);
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.STRING_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.STRING_ARRAY)
                    {
                        string[] oneStrArray = one.get<String[]>();
```

```
                        string[] twoStrArray = two.get<String[]>();
                        if (oneStrArray != null && twoStrArray != null)
                        {
                            return new FaultDetectionResult(oneStrArray.Length <=
twoStrArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid STRING_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.INT_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.INT_ARRAY)
                    {
                        int[] oneIntArray = one.get<int[]>();
                        int[] twoIntArray = two.get<int[]>();
                        if (oneIntArray != null && twoIntArray != null)
                        {
                            return new FaultDetectionResult(oneIntArray.Length <=
twoIntArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid INT_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.DECIMAL_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.DECIMAL_ARRAY)
                    {
                        Decimal[] oneArray = one.get<Decimal[]>();
                        Decimal[] twoArray = two.get<Decimal[]>();
                        if (oneArray != null && twoArray != null)
                        {
                            return new FaultDetectionResult(oneArray.Length <=
twoArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid DECIMAL_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.TIMESPAN:
                    if (two.ComputedType == FaultDetectionParamType.TIMESPAN)
                    {
                        return new FaultDetectionResult(one.get<TimeSpan>() <=
two.get<TimeSpan>());
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
            }
            return new FaultDetectionResult(false, "paramter 1 type (" + one.Type + "
== " + one.ComputedType + ") is invalid for SHORTER_EQUALS operation");
        }

    }
}
```

1.4.7.10 FaultDetectionOperationShorterThan.cs

Checks if one FaultDetectionParam (computed value) is shorter than (length) to a 2nd FaultDetectionParam (computed value). Valid FaultDetectionParamType combinations:

- STRING, STRING
- STRING_ARRAY, STRING_ARRAY
- INT_ARRAY, INT_ARRAY
- DECIMAL_ARRAY, DECIMAL_ARRAY
- TIMESPAN, TIMESPAN

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FaultDetection.Operations
{
    /// <summary>
    /// Represents a "Shorter Than (length)" operation:
    ///
    /// Examaple:
    /// evaluate("XZ","XXZ") will pass
    /// evaluate([1,2,3],[1,2,3]) will fail
    /// evaluate("RX","X") will fail
    /// </summary>
    class FaultDetectionOperationShorterThan : IFaultDetectionOperation
    {
        /// <summary>
        /// Checks if one FaultDetectionParam (computed value) is shorter than
(length) to a 2nd FaultDetectionParam (computed value)
        ///
        /// Valid FaultDetectionParamType combinations:
        ///     STRING, STRING
        ///     STRING_ARRAY, STRING_ARRAY
        ///     INT_ARRAY, INT_ARRAY
        ///     DECIMAL_ARRAY, DECIMAL_ARRAY
        ///     TIMESPAN, TIMESPAN
        /// </summary>
        /// <param name="one">a base FaultDetectionParam to compare</param>
        /// <param name="two">a FaultDetectionParam to compare to</param>
        /// <returns>FaultDetectionResult.Passed = True if 1st param shorter than
(length) to a 2nd param, false otherwise</returns>
        public FaultDetectionResult evaluate(FaultDetectionParam one,
FaultDetectionParam two)
        {
            // using switch since parameter types might be different.
            switch (one.ComputedType)
            {
                case FaultDetectionParamType.STRING:
                    if (two.ComputedType == FaultDetectionParamType.STRING)
                    {
                        return new FaultDetectionResult(one.get<String>().Length <
two.get<String>().Length);
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.STRING_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.STRING_ARRAY)
                    {
```

```
                        string[] oneStrArray = one.get<String[]>();
                        string[] twoStrArray = two.get<String[]>();
                        if (oneStrArray != null && twoStrArray != null)
                        {
                            return new FaultDetectionResult(oneStrArray.Length <
twoStrArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid STRING_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.INT_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.INT_ARRAY)
                    {
                        int[] oneIntArray = one.get<int[]>();
                        int[] twoIntArray = two.get<int[]>();
                        if (oneIntArray != null && twoIntArray != null)
                        {
                            return new FaultDetectionResult(oneIntArray.Length <
twoIntArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid INT_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.DECIMAL_ARRAY:
                    if (two.ComputedType == FaultDetectionParamType.DECIMAL_ARRAY)
                    {
                        Decimal[] oneArray = one.get<Decimal[]>();
                        Decimal[] twoArray = two.get<Decimal[]>();
                        if (oneArray != null && twoArray != null)
                        {
                            return new FaultDetectionResult(oneArray.Length <
twoArray.Length);
                        }
                        return new FaultDetectionResult(false, "paramter 1 (" +
one.Value + " == " + one.ComputedValue + ") or parameter 2 (" + two.Value + " == " +
two.ComputedValue + ") are invalid DECIMAL_ARRAYs");
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);
                case FaultDetectionParamType.TIMESPAN:
                    if (two.ComputedType == FaultDetectionParamType.TIMESPAN)
                    {
                        return new FaultDetectionResult(one.get<TimeSpan>() <
two.get<TimeSpan>());
                    }
                    return FaultDetectionResult.invalidParameterOperator(one, two,
this);

            }
            return new FaultDetectionResult(false, "paramter 1 type (" + one.Type + "
== " + one.ComputedType + ") is invalid for SHORTER_THAN operation");
        }

    }
}
```

2. Fault Detection Monitoring Service

The Monitoring service was split into 2 projects, a Windows Service project and Class Library Project. The Class Library is the Kernel of the service and is used to do all the operations the service was meant to do. The idea behind using a Kernel Class Library was to allow for easier and faster debugging by consuming it through a Windows Form application.

2.1 Fault Detection Monitoring Service Kernel

2.1.1 Kernel.cs

FGBFaultDetectionServiceKernel.Kernel, used to do all the operation the service was meant to do. The idea of a Kernel DLL allows for a "service" to be debugged as Windows Form application and saves a lot of development time.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Threading;
using System.Configuration;
using System.Net.Sockets;
using System.Reflection;
using FaultDetection.Environments;
using FaultDetection;
using FGBFaultDetectionServiceKernel.FGBWeb_TMC;
using FGBFaultDetectionServiceKernel.FGBWeb_General;
using FGBFaultDetectionServiceKernel.FGBWeb_TMCWeb;
using FaultDetection.Modules;
using FGBFaultDetectionServiceKernel.Loggers;
using System.IO;
using Newtonsoft.Json;

namespace FGBFaultDetectionServiceKernel
{
    /// <summary>
    /// FGBFaultDetectionService Kernel, used to do all the operation the service was
meant to do.
    ///
    /// The idea of a Kernel DLL allows for a "service" to be debugged as Windows Form
application and saves a lot of development time.
    /// </summary>
    public class Kernel
    {
        private const string WebServiceKeyname = "FaultDetection_WebserviceURL";
        private const string FaultDetectionKnowledgebaseKeyname =
"xFGBFaultDetection";

        private string _webserviceURL = "";
        private string _machineName = "";
        private string _ip = "";
        private string _hostName = "";
        private string _faultDetectionDBConnectionString = "";
        private bool _faultDetectionThreadNeeded = true;
        private bool _faultDetectionTriggered = false;
        private Thread FaultDetectionThread;

        private General _wsGeneral = null;
        private TMC _wsTMC = null;
        private TMCWeb _wsTMCWeb = null;
```

```
        private RegistryHelper _registryHelper;

        private int _faultDetectionIntervalValue = 3;
        private RegistryHelper.IntervalTypes _faultDetectionIntervalType =
RegistryHelper.IntervalTypes.UNKNOWN;
        public System.Timers.Timer FaultDetectionTimer;

        private static IFaultDetectionEnvironment _faultDetectionEnvironment = null;
        private static IFaultDetectionKnowledgebase _faultDetectionDB = null;
        private static FaultDetectionCaseCollection _faultDetectionBaseTMCCollection =
null;

        private static bool _sendNotifications = false;

        private MultiLogger _logger = new MultiLogger();
        /// <summary>
        /// Initalises the Kernel, loads the machine name, app.config information etc.
        /// </summary>
        /// <exception cref="Exception">thrown if there was any problem initialising
the Kernel</exception>
        public Kernel()
        {
            try
            {
                _machineName = System.Environment.MachineName;
                _wsGeneral = new FGBWeb_General.General();
                _wsTMC = new FGBWeb_TMC.TMC();
                _wsTMCWeb = new FGBWeb_TMCWeb.TMCWeb();
                _hostName = Dns.GetHostName();
                _ip = getIPAddress();

                ThreadPool.SetMaxThreads(700, 700);
                try
                {
                    _faultDetectionDBConnectionString =
ConfigurationManager.ConnectionStrings[FaultDetectionKnowledgebaseKeyname].ConnectionS
tring;
                    _webserviceURL =
ConfigurationManager.AppSettings[WebServiceKeyname];
                    updateWebServicesURLs();
                    this._logger.AddLogger(new WSLogger(_wsGeneral));
                    this._logger.AddLogger(new ConsoleLogger());
                }
                catch (Exception ex)
                {
                    logException(ex);
                    throw ex;
                }
            }
            catch (Exception ex)
            {
                logException(ex);
                throw ex;
            }
        }
        /// <summary>
        /// Starts the Kernel, loads the registry, creates an
IFaultDetectionKnowledgebase object and starts the Fault Detection schedule timer.
        /// </summary>
        /// <exception cref="Exception">thrown if there were any issues starting the
kernel</exception>
        public void Start()
```

```
        {
            log("Starting Kernel", ILoggerFlags.NORMAL);
            try
            {
                if (this.LoadRegistry())
                {
                    if (_faultDetectionDB == null) // might have been succesfully
loaded in the registry loading operation, no need to try again.
                    {
                        _faultDetectionDB =
FaultDetectionKnowledgebaseFactory.Create(_faultDetectionDBConnectionString);
                        if (_faultDetectionDB == null)
                        {
                            log("Failed creating IFaultDetectionKnowledgebase object
[" + _faultDetectionDBConnectionString + "]", ILoggerFlags.ERRORS);
                            this.Stop();
                            throw new Exception("Invalid FaultDetection DB Connection
String (" + _faultDetectionDBConnectionString + ")");
                        }
                    }
                    _logger.setFlags(this._registryHelper.LoggingFlags, true);
                    //FaultDetectionThreadStart(true);
                    FaultDetectionTimerStart();
                }
                else
                {
                    log("Failed to load Registry, shutting down",
ILoggerFlags.ERRORS);
                    this.Stop();
                    throw new Exception("Failed loading registry! [WebService URL: " +
_webserviceURL + "], please check App.config");
                }
            }
            catch(Exception ex) {
                logException(ex);
                this.Stop();
                throw ex;
            }
        }

        /// <summary>
        /// Stops the kernel.
        /// Terminates the Fault Detection Threads.
        /// Stops the Fault Detection schedule timer.
        /// </summary>
        public void Stop()
        {
            Stop(null);
        }
        /// <summary>
        /// Stops the kernel.
        /// Terminates the Fault Detection Threads.
        /// Stops the Fault Detection schedule timer.
        /// </summary>
        /// <param name="message">Stop message for logging</param>
        public void Stop(string message)
        {
            log("Stopping Kernel"+ (!string.IsNullOrEmpty(message) ? " ("+message+")"
: ""), ILoggerFlags.NORMAL);
            FaultDetectionThreadStop();
            FaultDetectionTimerStop();
            log("Kernel Stopped", ILoggerFlags.TRACE);
```

```
        }
        /// <summary>
        /// Stops, wait 1second and Starts the Kernel again.
        /// </summary>
        public void Restart()
        {
            this.Stop();
            Thread.Sleep(1000);
            this.Start();
        }

        #region settings
        /// <summary>
        /// Returns the IP address of the machine the kernel/service is running on.
        /// </summary>
        /// <returns></returns>
        private string getIPAddress()
        {
            IPHostEntry host;
            string localIP = "?";
            host = Dns.GetHostEntry(Dns.GetHostName());
            foreach (IPAddress ip in host.AddressList)
            {
                if (ip.AddressFamily == AddressFamily.InterNetwork)
                {
                    localIP = ip.ToString();
                }
            }
            return localIP;
        }
        /// <summary>
        /// Loads the registry from the FGBWeb_General Web Service.
        /// </summary>
        /// <returns>true if registry loaded without error, false otherwise</returns>
        /// <exception cref="Exception">thrown if there were any issues loading
Registry</exception>
        private bool LoadRegistry()
        {
           return LoadRegistry(_wsGeneral.RegistryGet(RegistryHelper.SoftwareId,
true));
        }
        /// <summary>
        /// Loads the registry the given FGBWeb_General Web Service Registry object.
        /// </summary>
        /// <param name="registry">FGBWeb_General.Registry return object to
load</param>
        /// <returns>true if registry loaded without error, false otherwise</returns>
        /// <exception cref="Exception">thrown if there were any issues loading
Registry</exception>
        private bool LoadRegistry(FGBWeb_General.Registry registry)
        {
            string method = MethodBase.GetCurrentMethod().Name;
            //log("Loading Registry", ILoggerFlags.TRACE, method);
            if (registry != null)
            {
                if (this._registryHelper == null)
                {
                    this._registryHelper = new RegistryHelper();
                }
                try
                {
                    if (registry != null)
```

```
                    {
                        if (Util.isSuccess(registry.Result))
                        {
                            //log("Got " + this._registry.Keys.Length + " Registry
Keys from WebService", ILoggerFlags.TRACE, method);
                            this._registryHelper.loadRegistry(registry);
                            //log("RegistryHelper Loaded Successfully",
ILoggerFlags.NORMAL, method);
                            if
(!this._webserviceURL.Equals(this._registryHelper.WebserviceURL))
                            {
                                this._webserviceURL =
this._registryHelper.WebserviceURL;
                                updateWebServicesURLs();
                                try
                                {
                                    _faultDetectionEnvironment =
loadFGBFCSEnvironment();
                                    if (_faultDetectionEnvironment == null)
                                    {
                                        throw new Exception("Unable to load FGBFCS
Enironment using the Registry Table's FaultDetection_WebserviceURL key: " +
this._webserviceURL);
                                    }
                                }
                                catch (Exception ex)
                                {
                                    throw ex;
                                }
                                this._faultDetectionDBConnectionString =
this._registryHelper.FaultDetectionDBConnectionString;
                                try
                                {
                                    _faultDetectionDB =
FaultDetection.FaultDetectionKnowledgebaseFactory.Create(this._faultDetectionDBConnect
ionString);
                                    if (_faultDetectionDB == null)
                                    {
                                        throw new Exception("Unable to crreate
IFaultDetectionKnowledgebase object from Registry Tables
FaultDetection_FaultDetectionDBConnectionString key: " +
this._faultDetectionDBConnectionString);
                                    }
                                }
                                catch (Exception ex)
                                {
                                    throw ex;
                                }
                                Configuration config =
ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
                                config.AppSettings.Settings[WebServiceKeyname].Value =
this._webserviceURL;
                                config.Save(ConfigurationSaveMode.Modified);
                                ConfigurationManager.RefreshSection("appSettings");

                            }
                            if (this._registryHelper.IntervalType !=
RegistryHelper.IntervalTypes.UNKNOWN || this._registryHelper.IntervalValue < 1)
                            {

                                if (this._faultDetectionIntervalType !=
this._registryHelper.IntervalType || this._registryHelper.IntervalValue !=
```

```
this._faultDetectionIntervalValue)
                                {

                                    this._faultDetectionIntervalType =
this._registryHelper.IntervalType;
                                    this._faultDetectionIntervalValue =
this._registryHelper.IntervalValue;
                                    if (this._faultDetectionIntervalType ==
RegistryHelper.IntervalTypes.DAILY && this._faultDetectionIntervalValue > 23)
                                    {
                                        throw new Exception("Invalid Interval Value ("
+ this._faultDetectionIntervalValue + ") For Daily Type Interval, value should be an
hour in the day, i.e. 0 (midnight) to 23 (11pm)");
                                    }
                                    FaultDetectionTimerResetIntrerval(true);

                                }
                            }
                            else
                            {
                                throw new Exception("Invalid Interval Type
("+this._registryHelper.IntervalType+") or Interval Value
("+this._registryHelper.IntervalValue+") Loaded from DB");
                            }

                            log("RegistryHelper Loaded", ILoggerFlags.INFO);
                            return true;
                        }
                    }

                }
                // catch the exception for logging and throw it again
                catch (Exception ex)
                {
                    logException(ex);
                    throw ex;
                }

            }
            return false;
        }
        #endregion

        #region general
        /// <summary>
        /// Updates all the underlying web services url with their appropriate asmx
files.
        /// </summary>
        private void updateWebServicesURLs()
        {
            _wsGeneral.Url = _webserviceURL + "general.asmx";
            _wsTMC.Url = _webserviceURL + "tmc.asmx";
            _wsTMCWeb.Url = _webserviceURL + "tmcweb.asmx";
        }
        /// <summary>
        /// Creates a FaultDetectionEnvironmentFGBFCS (IFaultDetectionEnvironment)
from the FGBWeb_General Web Service.
        /// </summary>
        /// <returns>FaultDetectionEnvironmentFGBFCS loaded from the FGBWeb_General
Web Service</returns>
        private FaultDetectionEnvironmentFGBFCS loadFGBFCSEnvironment()
        {
```

```
            try
            {
                FaultDetectionEnvironmentFGBFCS env = new
FaultDetectionEnvironmentFGBFCS();
                FGBWeb_General.FaultDetectionEnvironment fdenv =
_wsGeneral.FaultDetectionEnvironmentGet();
                foreach (FGBWeb_General.FaultDetectionEnvironmentAttribute attr in
fdenv.Attributes)
                {
                    env.Attributes.Add(attr.Name, new FaultDetectionParam(attr.Value,
true));
                }
                return env;
            }
            catch (Exception ex)
            {
                logException(ex);
                log("Failed Loading FGBFCS Environment: " + ex.Message,
ILoggerFlags.ERRORS);
                return null;
            }
        }
        #endregion

        #region logging
        /// <summary>
        /// Adds an ILogger to the MultiLogger of the Kernel.
        /// This is public to allow consumers of the Kernel DLL to add their own
ILoggers.
        /// </summary>
        /// <param name="logger"></param>
        public void AddLogger(ILogger logger)
        {
            if (this._logger != null)
            {
                ((MultiLogger)this._logger).AddLogger(logger);
            }

        }

        /// <summary>
        /// Logs an Exception as a JSON string/file to the "_exceptions" subfolder of
the application/service using this Kernel.
        /// </summary>
        /// <param name="ex">The exception to log</param>
        internal void logException(Exception ex)
        {
            try
            {
                try
                {
                    System.OutOfMemoryException oomex =
(System.OutOfMemoryException)ex;
                    if (oomex != null)
                    {
                        Restart();
                    }
                }
                catch
                {
                }
                string root =
```

```
Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
                string exceptions = root + @"\_exceptions";
                if (!System.IO.Directory.Exists(exceptions))
                {
                    System.IO.Directory.CreateDirectory(exceptions);
                }
                if (System.IO.Directory.Exists(exceptions))
                {
                    string filename = "exception_" +
DateTime.Now.ToString().Replace('/', '-').Replace(':', '.') + ".json";
                    var sw = new StreamWriter(exceptions + @"\" + filename, false);
                    sw.WriteLine(JsonConvert.SerializeObject(ex));
                    sw.Close();
                    sw.Dispose();
                }
                log("Exception: " + ex.Message + "\r\nStackTrace: " + ex.StackTrace,
ILoggerFlags.ERRORS);
            }
            catch (Exception inEx)
            {
                log("Log Exception Failed: " + inEx.Message, ILoggerFlags.ERRORS);
            }
        }
        /// <summary>
        /// Wrapper method for the MultiLogger.WriteLine.
        /// </summary>
        /// <param name="line">the line to log</param>
        /// <param name="flags">the ILoggerFlags of logged line</param>
        internal void log(string line, ILoggerFlags flags)
        {
            this.log(line, flags, this.GetType().ToString());
        }
        /// <summary>
        /// Wrapper method for the MultiLogger.WriteLine.
        /// </summary>
        /// <param name="line">the line to log</param>
        /// <param name="flags">the ILoggerFlags of logged line</param>
        /// <param name="source">Where the logging is comming from</param>
        internal void log(string line, ILoggerFlags flags, string source)
        {
            if (this._logger != null)
            {
                this._logger.WriteLine(line, flags, source);
            }
        }

        #endregion

        #region tmc
        /// <summary>
        /// Loads FaultDetectionModuleTMCX a given FGBWeb_TMC Web Service
TMCTillInfoFaultDetection
        /// </summary>
        /// <param name="tmc">the TMCTillInfoFaultDetection for the TMC to
load</param>
        /// <returns>FaultDetectionModuleTMCX representing the given
TMCTillInfoFaultDetection</returns>
        private static FaultDetectionModuleTMCX
loadTMCModuleFromTMCTillInfoFaultDetection(TMCTillInfoFaultDetection tmc)
        {
            FaultDetectionModuleTMCX moduleTMCX = new FaultDetectionModuleTMCX();
            moduleTMCX.Settings.Add("IsCashBackEnabled", new
```

```
FaultDetectionParam(tmc.IsCashBackEnabled.Equals("1").ToString(),
FaultDetectionParamType.BOOLEAN));
            moduleTMCX.Settings.Add("IsCashBackOnCashBackEnabled", new
FaultDetectionParam(tmc.IsCashBackOnCashBackEnabled.Equals("1").ToString(),
FaultDetectionParamType.BOOLEAN));
            moduleTMCX.Settings.Add("PosID", new FaultDetectionParam(tmc.PosID,
FaultDetectionParamType.STRING));
            moduleTMCX.Properties.Add("DB_IP", tmc.IP);
            moduleTMCX.Properties.Add("DB_MACAddress", tmc.MacAddress);
            moduleTMCX.Properties.Add("DB_LastOnline", tmc.LastOnline.ToString());
            return moduleTMCX;
        }
        /// <summary>
        /// Loads a Base FaultDetectionCaseCollection for FaultDetectionModuleTMCX
module.
        ///
        /// This collection is then copied (without results etc) for specific TMC
before the Fault Detection check is done.
        /// </summary>
        /// <returns>FaultDetectionCaseCollection for the TMC Module</returns>
        private FaultDetectionCaseCollection loadTMCFaultDetectionCaseCollection()
        {
            try
            {
                return _faultDetectionDB.getFaultDetectionCollection(new
FaultDetectionModuleTMCX());
            }
            catch (Exception ex)
            {
                logException(ex);
                return null;
            }
        }
        #endregion

        #region notifications
        /// <summary>
        /// Creates a FaultDetectionEmailNotifier notifier and sets its SMTP settings
from the Registry.
        /// Sends notifications for all the teams in FaultDetectionCaseCollection who
had at least 1 fault case detected.
        /// </summary>
        /// <param name="env">The environment to use for notification (i.e. so the
support teams would know which Venue the fault happend on etc.)</param>
        /// <param name="fdCollection">The executed FaultDetectionCaseCollection to
generate notifications from</param>
        /// <exception cref="ApplicationException">Thrown if the SMTP server failed
sending the Email</exception>
        /// <exception cref="Exception">Thrown if an underlying exception
occured</exception>
        private void sendNotifications(IFaultDetectionEnvironment env,
FaultDetectionCaseCollection fdCollection)
        {
            IFaultDetectionNotifier notifier = new
FaultDetection.Notifiers.FaultDetectionEmailNotifier(fdCollection);
            Dictionary<string, string> notifierSettings = new Dictionary<string,
string>(5);
            notifierSettings.Add("smtpServer", _registryHelper.SMTPServer);
            notifierSettings.Add("smtpUsername", _registryHelper.SMTPUsername);
            notifierSettings.Add("smtpPassword", _registryHelper.SMTPPassword);
            notifierSettings.Add("emailSubject", "FGB Fault Detection - Faults
Detected at " + env.getAttribute("VenueShortName").Value);
```

```
            notifier.loadSettings(notifierSettings);
            notifier.sendNotifications();
        }
        #endregion

        #region Fault Detection
        /// <summary>
        /// Starts the Fault Detection schedule Timer
        /// </summary>
        private void FaultDetectionTimerStart()
        {
            if (this.FaultDetectionTimer == null)
            {
                if (FaultDetectionCalculateInterval(true) > 0) {
                    this.FaultDetectionTimer = new System.Timers.Timer();
                    this.FaultDetectionTimer.Interval =
FaultDetectionCalculateInterval(true);
                    this.FaultDetectionTimer.Elapsed += FaultDetectionEvent;
                    this.FaultDetectionTimer.Enabled = true;
                    this.FaultDetectionTimer.Start();

                    log("FaultDetectionTimer Started, Next Itteration:
"+DateTime.Now.AddMilliseconds(this.FaultDetectionTimer.Interval),
ILoggerFlags.NORMAL);
                }
            }
        }
        /// <summary>
        /// Stops the Fault Detection schedule Timer
        /// </summary>
        private void FaultDetectionTimerStop()
        {
            if (this.FaultDetectionTimer != null)
            {
                this.FaultDetectionTimer.Enabled = false;
                this.FaultDetectionTimer.Stop();

                log("FaultDetectionTimer Stopped", ILoggerFlags.NORMAL);
            }
        }
        /// <summary>
        /// Calculate the FaulDetection schedule timer interval based on the Registery
Interval Type and Value.
        /// </summary>
        /// <returns>the inverval value in milliseconds</returns>
        private double FaultDetectionCalculateInterval()
        {
            return FaultDetectionCalculateInterval(false);
        }
        /// <summary>
        /// Calculate the FaulDetection schedule timer interval based on the Registery
Interval Type and Value.
        /// </summary>
        /// <param name="first">Whether or not this is the first itteration of the
timer. (for Interval Type DAILY, the Inteval Value represents an hour of the
day)</param>
        /// <returns>the inverval value in milliseconds</returns>
        private double FaultDetectionCalculateInterval(bool first)
        {
            switch (this._faultDetectionIntervalType)
            {
                case RegistryHelper.IntervalTypes.DAILY:
```

```
                    if (first)
                    {
                        int intervalHrs = 0;
                        if (DateTime.Now.Hour < _faultDetectionIntervalValue)
                        {
                            intervalHrs = (_faultDetectionIntervalValue -
DateTime.Now.Hour) * (3600000) - ((DateTime.Now.Minute * 60000));
                        }
                        else
                        {
                            intervalHrs = ((24 - DateTime.Now.Hour) +
_faultDetectionIntervalValue) * (3600000) - ((DateTime.Now.Minute * 60000));
                        }
                        return intervalHrs;
                    }
                    return 86400000; // 24 hours in milliseconds.
                case RegistryHelper.IntervalTypes.HOURLY:
                    return this._faultDetectionIntervalValue * 3600000; // 1 hours in
milliseconds.
                case RegistryHelper.IntervalTypes.MINUTE:
                    return this._faultDetectionIntervalValue * 60000; // 1 minute in
milliseconds.
                default:
                    return 0;
            }
        }
        /// <summary>
        /// Resets the Fault Detection schedule Timer with a new interval
        /// </summary>
        /// <param name="first">Whether or not this is the first itteration of the
timer. (for Interval Type DAILY, the Inteval Value represents an hour of the
day)</param>
        private void FaultDetectionTimerResetIntrerval(bool first)
        {
            if (FaultDetectionTimer != null)
            {
                double interval = FaultDetectionCalculateInterval(first);
                if (FaultDetectionTimer.Interval != interval)
                {
                    FaultDetectionTimer.Interval = interval;
                    FaultDetectionTimer.Stop();
                    FaultDetectionTimer.Start();
                }
            }
        }
        /// <summary>
        /// An callback method used by the  Fault Detection schedule Timer when the
interval expires (i.e. a fault detection check should be executed)
        /// </summary>
        /// <param name="sender">the object executing this call back</param>
        /// <param name="e">the even arguments passed by the sender</param>
        private void FaultDetectionEvent(object sender, EventArgs e)
        {
            // if Interval Type is Daily, reset intveral to 24hours.
            if (this._faultDetectionIntervalType ==
RegistryHelper.IntervalTypes.DAILY)
            {
                FaultDetectionTimerResetIntrerval(false);
            }
            // make sure the FaulDetectionThread is started and alive.
            FaultDetectionThreadStart(true);
            log("FaultDetection Timer Event Triggered (Next Itteration: " +
```

```
DateTime.Now.AddMilliseconds(this.FaultDetectionTimer.Interval) + ")",
ILoggerFlags.INFO);
            if (FaultDetectionThread.IsAlive)
            {
                // sets the fault detection trigged to true, to be picked up by the
FaulDetectionThread and execute a Fault Detection check on a seperate thread.
                _faultDetectionTriggered = true;

            }
        }
        /// <summary>
        /// Starts the Fault Detection Thread
        /// </summary>
        /// <param name="needed">wehter or not the thread is needed. The thread will
only stay alive as long as its needed</param>
        private void FaultDetectionThreadStart(bool needed)
        {
            if (FaultDetectionThread == null || FaultDetectionThread.ThreadState ==
ThreadState.Stopped)
            {
                FaultDetectionThread = new Thread(new
ThreadStart(FaultDetectionPresistentThread));
                FaultDetectionThread.IsBackground = true;
                _faultDetectionThreadNeeded = needed;
                FaultDetectionThread.Start();

                log("FaultDetectionThread Started", ILoggerFlags.NORMAL);
            }

        }
        /// <summary>
        /// Stops the Fault Detection Thread by settings the Fault Detection Needed to
false and Fault Detection Triggered to false.
        /// </summary>
        private void FaultDetectionThreadStop()
        {
            _faultDetectionThreadNeeded = false;
            _faultDetectionTriggered = false;
            log("Stopped FaultDetectionThread " + (FaultDetectionThread != null ? " ("
+ FaultDetectionThread.ThreadState + ")" : ""), ILoggerFlags.NORMAL);
            /*if (FaultDetectionThread != null || FaultDetectionThread.ThreadState !=
ThreadState.Stopped)
            {
                FaultDetectionThread.Join();
                log("FaultDetectionThread Stopped (Join)", ILoggerFlags.NORMAL);
            }*/
        }
        /// <summary>
        /// Represents a presistent Fault Detection Thread that stays alive as long as
_faultDetectionThreadNeeded is true
        /// Listens to see if _faultDetectionTriggered is true, one it becomes true, a
Fault Detection check is executed and _faultDetectionTriggered is reset to false.
        /// </summary>
        private void FaultDetectionPresistentThread()
        {
            string method = MethodBase.GetCurrentMethod().Name;
            try
            {
                while (_faultDetectionThreadNeeded)
                {
                    if (_faultDetectionTriggered)
                    {
```

```
                        _faultDetectionTriggered = false;
                        //_fileReaderThreadNeeded = false;
                        executeFaultDetection();
                    }
                    Thread.Sleep(100);
                }
            }
            catch (Exception ex)
            {
                logException(ex);
            }
        }
        /// <summary>
        /// Executes a fault detection tests.
        /// Reloads the Environment from FGBWeb_General Web Service
        /// Reloads the base FaultDetectionCaseCollection for the TMC module from
Fault Detection Knowledge-base
        ///
        /// Gets a list of all the TMCs from FGBWeb_TMC Web Service, starts a fault
detection check for each one on a seperate thread using a ThreadPool.
        /// </summary>
        private void executeFaultDetection()
        {
            _faultDetectionEnvironment = loadFGBFCSEnvironment(); // reload
environemnt to get the latest info/settings
            if (_faultDetectionEnvironment != null)
            {
                _faultDetectionBaseTMCCollection =
loadTMCFaultDetectionCaseCollection(); // reload the base fault detection case
collection to get the latest version
                _sendNotifications = this._registryHelper.SendNotifications; //
check/set whether or not to issue notifications based on the registry settings.
                // if fault detection db and base collection are okay, start...
                if (_faultDetectionDB != null && _faultDetectionBaseTMCCollection !=
null && _faultDetectionBaseTMCCollection.List != null &&
_faultDetectionBaseTMCCollection.List.Count > 0)
                {
                    try
                    {
                        // get the latest list of TMCs from the FGBWeb_TMC WebService,
and loop through it.
                        foreach (TMCTillInfoFaultDetection tmc in
_wsTMC.TMCGetFullDetailsList(null))
                        {
                            // if the kerenl is stopped midway through, dont
create/start new fault detection checks.
                            if (!_faultDetectionThreadNeeded)
                            {
                                break;
                            }
                            // start a fault detection test on each tmc in a seperate
thread.
                            ThreadPool.QueueUserWorkItem(new
WaitCallback(executeFaultDetectionOnTMCThread), tmc);
                        }
                    }
                    catch (Exception ex)
                    {
                        logException(ex);
                        log("Exception executeFaultDetection: " + ex.Message,
ILoggerFlags.ERRORS);
                    }
```

```
                }
            }
            else
            {
                log("Execution Skipped, Environment Was Not Loaded",
ILoggerFlags.ERRORS | ILoggerFlags.INFO);
            }
        }
        /// <summary>
        /// Wrapper for executeFaultDetectionOnTMC, converts the given senderInfo
object to a TMCTillInfoFaultDetection object and executes a fault detection check.
        /// </summary>
        /// <param name="senderInfo">a senderInfo object representing a
TMCTillInfoFaultDetection</param>
        private void executeFaultDetectionOnTMCThread(object senderInfo)
        {
            try
            {
                TMCTillInfoFaultDetection tmc = (TMCTillInfoFaultDetection)senderInfo;
                if (tmc != null)
                {
                    executeFaultDetectionOnTMC(tmc);
                }
            }
            catch
            {
            }
        }

        /// <summary>
        /// Executes a Fault Detection check on a TMC from FGBWeb_TMC Web Service.
        ///
        /// Check will be executed only if the TMC has been online during the Registry
MaxTimeInMinutesSinceLastOnline threshold (if its not online, we can get its current
state anyway).
        /// </summary>
        /// <param name="tmc">TMCTillInfoFaultDetection representing the TMC to
execute fault detection on</param>
        private void executeFaultDetectionOnTMC(TMCTillInfoFaultDetection tmc)
        {
            if (_faultDetectionDB != null && _faultDetectionEnvironment != null && tmc
!= null && _faultDetectionBaseTMCCollection != null &&
_faultDetectionBaseTMCCollection.List != null &&
_faultDetectionBaseTMCCollection.List.Count > 0)
            {
                string tmcCaption = tmc.TillName + " / " + tmc.PosID + " (" + tmc.IP +
" / " + tmc.MacAddress + ")";
                log("executeFaultDetectionOnTMC Called On " + tmcCaption + " - Last
Online: " + tmc.LastOnline + " [Threshold: " + (DateTime.Now.AddMinutes(0 -
this._registryHelper.MaxTimeInMinutesSinceLastOnline))+"]", ILoggerFlags.INFO);
                FaultDetectionResult result = new FaultDetectionResult();
                FaultDetectionExecution execution = null;
                // check if the tmc has been online since
MaxTimeInMinutesSinceLastOnline
                if (tmc.LastOnline >= (DateTime.Now.AddMinutes(0 -
this._registryHelper.MaxTimeInMinutesSinceLastOnline)))
                {
                    log("Starting Fault Detection On "+tmcCaption,
ILoggerFlags.TRACE);
                    FaultDetection.Modules.FaultDetectionModuleTMCX moduleTMCX =
loadTMCModuleFromTMCTillInfoFaultDetection(tmc); // create a IFaultDetectionModule
from the TMCTillInfoFaultDetection object.
```

```
                    TMCResponse tmcResponse = getTMCResponse(tmc, tmcCaption, result);
                    if (tmcResponse != null)
                    {
                        string tmcStatus = loadTMCStatusFromHex(tmcCaption,
tmcResponse.Message.SpareStr1, result);
                        if (!string.IsNullOrEmpty(tmcStatus))
                        {
                            if (loadTMCStatus(ref moduleTMCX, tmcCaption,
tmcStatus,result) >= 0) // if any attributes were loaded, begin fault detection check.
                            {
                                try
                                {
                                    log("Executing Fault Detection on "+tmcCaption,
ILoggerFlags.INFO | ILoggerFlags.TRACE);
                                    FaultDetectionCaseCollection fdCollection =
_faultDetectionBaseTMCCollection.Copy(moduleTMCX); // instead of making a db call, we
just copy the base collection.
                                    FaultDetectionEngine engine = new
FaultDetectionEngine(_faultDetectionEnvironment); // initialise an engine object with
the current environment
                                    result = engine.execute(ref fdCollection); //
execute fault detection
                                    execution =
_faultDetectionDB.saveFaultDetectionExecution(fdCollection); // save result to the
Fault Detection Knowlege-base
                                    log("Executing Fault Detection on "+tmcCaption+"
Completed -> "+result.Passed+ (!result.Passed && result.Error.Length > 0 ? "
("+result.Error+")" : ""), ILoggerFlags.INFO);
                                    if (!result.Passed && _sendNotifications) // if
one of the fault cases failed (i.e. fault detected) and send notification is true,
send notifications.
                                    {
                                        try
                                        {

sendNotifications(_faultDetectionEnvironment, fdCollection);
                                            log("Notifications sent",
ILoggerFlags.INFO | ILoggerFlags.TRACE);
                                        }
                                        catch (Exception ex)
                                        {
                                            logException(ex);
                                            log("Failed sending Notifications: " +
ex.Message, ILoggerFlags.ERRORS);
                                        }
                                    }
                                }
                                catch(Exception ex)
                                {
                                    logException(ex);
                                    log("Exception Executing Fault Detection on " +
tmcCaption+": "+ex.Message, ILoggerFlags.ERRORS | ILoggerFlags.TRACE);
                                }
                            }
                        }
                        else
                        {
                            result.Error = "TMC Status String is Empty [" + tmcStatus
+ "]";
                            result.Executed = false;
                            log("Skipping Fault Detection on " + tmcCaption + " ->
Empty Status (" + tmcStatus + ")", ILoggerFlags.INFO | ILoggerFlags.WARNING);
```

```
                        }
                    }
                }
                else
                {
                    result.Error = "Skipped - TMC Not Online "+(tmc != null ? "(Last
Online: "+tmc.LastOnline+")" : "");
                    log("Skipping Fault Detection on " + tmcCaption + " -> Not Online
(Last Online: " + tmc.LastOnline + " / Threshold:
"+_registryHelper.MaxTimeInMinutesSinceLastOnline+"Minutes)", ILoggerFlags.INFO |
ILoggerFlags.WARNING);
                    result.Executed = false;
                }
                try
                {
                    // attach the fault detection execution result to the tmc in the
enviornment database.
                    _wsTMC.TMCSaveFaultDetectionResult(tmc.MacAddress,
(result.Executed ? (result.Passed ? 0 : 1) : -1), result.Error, (execution != null ?
execution.ID : -1));
                    log("TMC Result Saved", ILoggerFlags.TRACE);
                }
                catch(Exception ex)
                {
                    logException(ex);
                    log("Failing Saving TMC Result on " + tmcCaption + ": " +
ex.Message, ILoggerFlags.TRACE | ILoggerFlags.ERRORS);
                }
            }
        }
        /// <summary>
        /// Converts the Hex string returned by the Web Service to a textual string.
        /// </summary>
        /// <param name="tmcCaption">the tmc caption to use for logging</param>
        /// <param name="tmcResponseHexString">the returenx hex string from the web
service</param>
        /// <param name="result">the FaultDetectionResult to update if converation
worked or not</param>
        /// <returns>a JSON state string convereted from the Hex string
given</returns>
        private string loadTMCStatusFromHex(string tmcCaption, string
tmcResponseHexString, FaultDetectionResult result)
        {
            string tmcStatus = null;
            try
            {
                tmcStatus = FaultDetectionUtil.HexToString(tmcResponseHexString); //
convert the Hex string returned by the Web Service to a textual string.
                log("Converted  " + tmcCaption + " Status to String",
ILoggerFlags.TRACE);
            }
            catch (Exception ex)
            {
                result.Error = "Exception Converting TMC Status from Web Service: " +
ex.Message;
                result.Executed = false;
                tmcStatus = null;
                logException(ex);
                log("Exception Converting " + tmcCaption + " Status to String: " +
ex.Message, ILoggerFlags.TRACE | ILoggerFlags.ERRORS);
            }
            return tmcStatus;
```

```
        }
        /// <summary>
        /// Loads the given TMC Module with the give JSON State string.
        /// </summary>
        /// <param name="moduleTMCX">the TMC module to load the JSON State to</param>
        /// <param name="tmcCaption">the tmc caption to use for logging</param>
        /// <param name="tmcStatus">the tmc state JSON string to load</param>
        /// <param name="result">the FaultDetectionResult to update if loading worked
or not</param>
        /// <returns>the number of attributes loaded</returns>
        private int loadTMCStatus(ref FaultDetection.Modules.FaultDetectionModuleTMCX
moduleTMCX, string tmcCaption, string tmcStatus, FaultDetectionResult result)
        {
            int loaded = -1;
            try
            {
                loaded = moduleTMCX.loadStatus(tmcStatus); // load the TMC JSON State
String before executing Fault Detection
                log("Loaded  " + tmcCaption + " Status [" + loaded + " Attributes
Loaded]", ILoggerFlags.TRACE);
            }
            catch (Exception ex)
            {
                result.Error = "Exception Loading TMC Status: " + ex.Message;
                result.Executed = false;
                loaded = -1;
                logException(ex);
                log("Exception Loading " + tmcCaption + " Status: " + ex.Message,
ILoggerFlags.TRACE | ILoggerFlags.ERRORS);
            }
            return loaded;
        }
        /// <summary>
        /// Gets a TMCResponse (including it the TMC State String in Hex) from the
FGBWeb_TMC Web Service, using the IP of the give TMCTillInfoFaultDetection object.
        /// </summary>
        /// <param name="tmc">TMCTillInfoFaultDetection to get the State String
for</param>
        /// <param name="tmcCaption">the tmc caption used for logging</param>
        /// <param name="result">the FaultDetectionResult to update if getting a
response worked or not</param>
        /// <returns>the TMCResponse returned by the web service</returns>
        private TMCResponse getTMCResponse(TMCTillInfoFaultDetection tmc, string
tmcCaption, FaultDetectionResult result)
        {
            TMCResponse tmcResponse = null;
            try
            {
                tmcResponse = _wsTMCWeb.GetStatusOfTMCByIP(tmc.IP); // get the TMC
JSON State string using the Web Service (since sometimes the computer running this
service wont have direct access to the TMCs)
                log("Got " + tmcCaption + " Status", ILoggerFlags.TRACE);
            }
            catch (Exception ex)
            {
                result.Error = "Exception getting TMC Status: " + ex.Message;
                result.Executed = false;
                tmcResponse = null;
                logException(ex);
                log("Exception getting " + tmcCaption + " Status: " + ex.Message,
ILoggerFlags.TRACE | ILoggerFlags.ERRORS);
            }
```

```
            return tmcResponse;
        }
        #endregion
    }
}
```

2.1.2 Classes\ILogger.cs

Represents a uniform Logging mechanism for the Kernel.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FGBFaultDetectionServiceKernel
{
    /// <summary>
    /// Represents a uniform Logging mechanism for the Kernel.
    /// </summary>
    public interface ILogger
    {
        /// <summary>
        /// returns the name of the ILogger
        /// </summary>
        /// <returns>the name of the ILogger</returns>
        string GetName();
        /// <summary>
        /// Sets the logging flags of the ILogger.
        /// </summary>
        /// <param name="flags">ILoggerFlags representing which events to log and
which to ignore</param>
        void setFlags(ILoggerFlags flags);
        /// <summary>
        /// Write a line to the log
        /// </summary>
        /// <param name="line">the line to log</param>
        void WriteLine(string line);
        /// <summary>
        /// Write a line to the log with certain flags
        /// </summary>
        /// <param name="line">the line to log</param>
        /// <param name="flags">the flags to use for the line</param>
        void WriteLine(string line, ILoggerFlags flags);
        /// <summary>
        /// Write a line to the log with certain flags and indication where it came
from.
        /// </summary>
        /// <param name="line">the line to log</param>
        /// <param name="flags">the flags to use for the line</param>
        /// <param name="source">where the line came from (method, class, operation,
etc.)</param>
        void WriteLine(string line, ILoggerFlags flags, string source);
    }

    [Flags]
    public enum ILoggerFlags : int
    {
        NORMAL = 1,

        ERRORS = 2,
        INFO = 4,
        WARNING = 8,
```

```
        EVENTLOG = NORMAL,

        TRACE = 4096,

        DEBUG = NORMAL | ERRORS | INFO | WARNING | EVENTLOG | TRACE
    }
}
```

2.1.3 Classes\IMultiLogger.cs

Represents a collection of ILoggers.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FGBFaultDetectionServiceKernel
{
    /// <summary>
    /// Represents a collection of ILoggers.
    /// </summary>
    public interface IMultiLogger : ILogger
    {
        /// <summary>
        /// Adds an ILogger to the collection
        /// </summary>
        /// <param name="logger">the ILogger to add</param>
        void AddLogger(ILogger logger);
        /// <summary>
        /// Sets the MultiLogger Flags
        /// </summary>
        /// <param name="flags">the ILoggerFlags to use</param>
        /// <param name="includeLoggers">if true, will also set these flags for all
the ILoggers in the collection</param>
        void setFlags(ILoggerFlags flags, bool includeLoggers);
        /// <summary>
        /// Sets all the ILoggers flags in the collection to the given flags.
        /// </summary>
        /// <param name="flags">the ILoggerFlags to use</param>
        void setLoggersFlags(ILoggerFlags flags);
        /// <summary>
        /// Write a Line using a sepecific ILogger by its name
        /// </summary>
        /// <param name="loggerName">the ILogger name to use</param>
        /// <param name="line">the line to log</param>
        void WriteLine(string loggerName, string line);
        /// <summary>
        /// Write a Line using a sepecific ILogger by its name, using certain flags.
        /// </summary>
        /// <param name="loggerName">the ILogger name to use</param>
        /// <param name="line">the line to log</param>
        /// <param name="flags">the line flags to use</param>
        void WriteLine(string loggerName, string line, ILoggerFlags flags);
        /// <summary>
        /// Write a Line using a sepecific ILogger by its name, using certain flags
and specifying a source.
        /// </summary>
        /// <param name="loggerName">the ILogger name to use</param>
        /// <param name="line">the line to log</param>
        /// <param name="flags">the line flags to use</param>
        /// <param name="source">the source of where the line came from (method,
```

```
class, operation, etc.)</param>
        void WriteLine(string loggerName, string line, ILoggerFlags flags, string
source);
    }
}
```

2.1.4 Classes\RegistryHelper.cs

A Helper class for Registry Values. Enables easier access to Registry Keys (loaded from the Environment (FGBFCS) Database using Fortress Web Services).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using FGBFaultDetectionServiceKernel.FGBWeb_General;

namespace FGBFaultDetectionServiceKernel
{
    /// <summary>
    /// An Helper class for Registry Values.
    /// Enables easier access to Registry Keys.
    /// </summary>
    public class RegistryHelper
    {
        public const int SoftwareId = 148545;
        private Registry _registry = new Registry();

        private IntervalTypes _intervalType = IntervalTypes.DAILY;
        private int _intervalValue = 3;

        private int _maxTimeInMinutesSinceLastOnline = 60;

        private ILoggerFlags _loggingFlags = ILoggerFlags.NORMAL;

        private string _webserviceURL = "";
        private string _faultDetectionDBConnectionString = "";

        private string _smtpServer, _smtpUsername, _smtpPassword;
        private bool _sendNotifications;

        public enum IntervalTypes
        {
            UNKNOWN = 0,

            DAILY = 1,
            HOURLY = 2,
            MINUTE = 3
        }
        public bool SendNotifications
        {
            get { return this._sendNotifications; }
            set
            {
                SaveKey(KeyNames.SendNotifications, (this._sendNotifications ? 1 :
0));
                this._sendNotifications = value;
            }
        }
        public string SMTPServer {
            get { return this._smtpServer; }
            set { SaveKey(KeyNames.SMTPServer,value); this._smtpServer = value; }
```

```
        }
        public string SMTPUsername {
            get { return this._smtpUsername; }
            set { SaveKey(KeyNames.SMTPUsername,value); this._smtpUsername = value; }
        }
        public string SMTPPassword {
            get { return this._smtpPassword; }
            set { SaveKey(KeyNames.SMTPPassword, value); this._smtpPassword = value; }
        }
        public IntervalTypes IntervalType
        {
            get { return this._intervalType; }
            set {
                SaveKey(KeyNames.IntervalType, (int)value);
                this._intervalType = value;
            }
        }
        public int IntervalValue
        {
            get { return this._intervalValue; }
            set
            {
                SaveKey(KeyNames.IntervalValue, (int)value);
                this._intervalValue = value;
            }
        }

        public ILoggerFlags LoggingFlags
        {
            get { return this._loggingFlags; }
            set
            {
                SaveKey(KeyNames.LoggingFlags, (int)value);
                this._loggingFlags = value;
            }
        }

        public int MaxTimeInMinutesSinceLastOnline
        {
            get { return this._maxTimeInMinutesSinceLastOnline; }
            set
            {
                SaveKey(KeyNames.MaxTimeInMinutesSinceLastOnline, (int)value);
                this._maxTimeInMinutesSinceLastOnline = value;
            }
        }
        public string WebserviceURL
        {
            get { return this._webserviceURL; }
            set
            {
                SaveKey(KeyNames.WebserviceURL, value);
                this._webserviceURL = value;
            }
        }
        public string FaultDetectionDBConnectionString
        {
            get { return this._faultDetectionDBConnectionString; }
            set
            {
                SaveKey(KeyNames.FaultDetectionDBConnectionString, value);
                this._faultDetectionDBConnectionString = value;
```

```
            }
        }
        private struct KeyNames
        {
            public const string IntervalType = "FaultDetection_IntervalType";
            public const string IntervalValue = "FaultDetection_IntervalValue";

            public const string MaxTimeInMinutesSinceLastOnline =
"FaultDetection_MaxTimeInMinutesSinceLastOnline";

            public const string LoggingFlags = "FaultDetection_LoggingFlags";

            public const string WebserviceURL = "FaultDetection_WebserviceURL";
            public const string FaultDetectionDBConnectionString =
"FaultDetection_FaultDetectionDBConnectionString";

            public const string SMTPServer = "FaultDetection_SMTPServer";
            public const string SMTPUsername = "FaultDetection_SMTPUsername";
            public const string SMTPPassword = "FaultDetection_SMTPPassword";
            public const string SendNotifications =
"FaultDetection_SendNotifications";
        }
        /// <summary>
        /// Creates a new empty Registry Helper
        /// </summary>
        public RegistryHelper()
        {

        }
        /// <summary>
        /// Creates a Registry Helper and loads the given Registry object
        /// </summary>
        /// <param name="registry">a registry object to load</param>
        public RegistryHelper(Registry registry)
        {

        }
        /// <summary>
        /// Loads/Reloads a Registry object to this RegistryHelper
        /// </summary>
        /// <param name="registry">a registry object to load</param>
        public void loadRegistry(Registry registry)
        {
            this._registry = registry;
            foreach (RegistryKey key in _registry.Keys) {
                if (key.KeyName.Equals(KeyNames.IntervalType))
                {
                    try
                    {
                        if (Enum.IsDefined(typeof(IntervalTypes), key.NumValue))
                        {
                            this._intervalType =
(IntervalTypes)Enum.ToObject(typeof(IntervalTypes), key.NumValue);
//(IntervalTypes)key.NumValue;
                        }
                        else
                        {
                            this._intervalType = IntervalTypes.UNKNOWN;

                        }
                    }
                    catch
```

```
                    {
                        this._intervalType = IntervalTypes.UNKNOWN;
                    }
                }
                else if (key.KeyName.Equals(KeyNames.IntervalValue))
                {
                    this._intervalValue = key.NumValue;
                }
                else if (key.KeyName.Equals(KeyNames.WebserviceURL))
                {
                    this._webserviceURL = key.StrValue;
                }
                else if
(key.KeyName.Equals(KeyNames.FaultDetectionDBConnectionString))
                {
                    this._faultDetectionDBConnectionString = key.StrValue;
                }
                else if (key.KeyName.Equals(KeyNames.SMTPServer))
                {
                    this._smtpServer = key.StrValue;
                }
                else if (key.KeyName.Equals(KeyNames.SMTPUsername))
                {
                    this._smtpUsername = key.StrValue;
                }
                else if (key.KeyName.Equals(KeyNames.SMTPPassword))
                {
                    this._smtpPassword = key.StrValue;
                }
                else if (key.KeyName.Equals(KeyNames.SendNotifications))
                {
                    this._sendNotifications = (key.NumValue == 1);
                }
                else if (key.KeyName.Equals(KeyNames.MaxTimeInMinutesSinceLastOnline))
                {
                    if (key.NumValue >= 0)
                    {
                        this._maxTimeInMinutesSinceLastOnline = key.NumValue;
                    }
                }
                else if (key.KeyName.Equals(KeyNames.LoggingFlags))
                {
                    try
                    {
                        this._loggingFlags = (ILoggerFlags)key.NumValue;
                    }
                    catch
                    {
                        this._loggingFlags = ILoggerFlags.NORMAL;
                    }
                }
            }
        }

        /// <summary>
        /// Updates Registry Key String Value
        /// </summary>
        /// <param name="keyName">the key name of the key to update</param>
        /// <param name="numValue">the new string value</param>
        private void SaveKey(string keyName, string stringValue)
        {
            RegistryKey key = GetKey(keyName);
```

```
            if (key != null)
            {
                key.StrValue = stringValue;
            }
        }
        /// <summary>
        /// Updates Registry Key Num Value
        /// </summary>
        /// <param name="keyName">the key name of the key to update</param>
        /// <param name="numValue">the new num value</param>
        private void SaveKey(string keyName, int numValue)
        {
            RegistryKey key = GetKey(keyName);
            if (key != null)
            {
                key.NumValue = numValue;
            }
        }
        /// <summary>
        /// Gets a Registry Key from the loaded Registry.
        /// </summary>
        /// <param name="keyName">the name of the key to get</param>
        /// <returns>a registry key</returns>
        public RegistryKey GetKey(string keyName)
        {
            foreach (RegistryKey key in _registry.Keys)
            {
                if (key.KeyName.Equals(keyName))
                {
                    return key;
                }
            }
            return null;
        }
        /// <summary>
        /// Shortcut to get a key Num Value
        /// </summary>
        /// <param name="keyName">the name of the key to get</param>
        /// <returns>an int represeting the Num Value of the registry key</returns>
        public int GetNumValue(string keyName)
        {
            RegistryKey key = GetKey(keyName);
            if (key != null)
            {
                return key.NumValue;
            }
            return -1;
        }
        /// <summary>
        /// Shortcut to get a key String Value
        /// </summary>
        /// <param name="keyName">the name of the key to get</param>
        /// <returns>a string represeting the String Value of the registry
key</returns>
        public string GetStringValue(string keyName)
        {
            RegistryKey key = GetKey(keyName);
            if (key != null)
            {
                return key.StrValue;
            }
            return null;
```

```
        }
        /// <summary>
        /// Saves the registry to the database using FGBWeb_General Web Service
        /// </summary>
        /// <param name="_ws">FGBWeb_General Web Service to use to save the
registry</param>
        /// <returns>true if save worked, false otherwise</returns>
        public bool Save(FGBWeb_General.General _ws)
        {
            if (_ws != null)
            {
                this._registry = _ws.RegistrySave(this._registry);
                return Util.isSuccess(this._registry.Result);
            }
            return false;
        }

    }
}
```

2.1.4 Classes\Util.cs

A Utility Class containing general methods/functions. Used to minimise code duplication.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using FGBFaultDetectionServiceKernel.FGBWeb_TMC;

namespace FGBFaultDetectionServiceKernel
{
    /// <summary>
    /// A Utility Class containing general methods/functions
    /// Used to minimise code duplication.
    /// </summary>
    public static class Util
    {
        /// <summary>
        /// Checks if a given SqlResult is successful or not.
        /// </summary>
        /// <param name="res">the SqlResul to check</param>
        /// <returns>true if the SqlResult had no errors, false otherwise</returns>
        public static bool
isSuccess(FGBFaultDetectionServiceKernel.FGBWeb_TMC.SqlResult res)
        {
            return (res != null && res.ErrorCode != null &&
res.ErrorCode.Equals("S"));
        }
        /// <summary>
        /// Checks if a given SqlResult is successful or not.
        /// </summary>
        /// <param name="res">the SqlResul to check</param>
        /// <returns>true if the SqlResult had no errors, false otherwise</returns>
        public static bool
isSuccess(FGBFaultDetectionServiceKernel.FGBWeb_General.SqlResult res)
        {
            return (res != null && res.ErrorCode != null &&
res.ErrorCode.Equals("S"));
        }
    }
}
```

2.1.5 Loggers\ConsoleLogger.cs

ILogger implementation that logs to the Visual Studio Console

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using System.Threading;

namespace FGBFaultDetectionServiceKernel.Loggers
{
    /// <summary>
    /// Logs to the Visual Studio Console
    /// </summary>
    class ConsoleLogger : ILogger
    {
        private string name = "ConsoleLogger";
        private ILoggerFlags _flags = ILoggerFlags.NORMAL;

        #region ILogger Members

        public void setFlags(ILoggerFlags flags)
        {
            this._flags = flags;
        }
        public string GetName()
        {
            return this.name;
        }

        public void WriteLine(string line)
        {
            WriteLine(line, ILoggerFlags.NORMAL);
        }
        public void WriteLine(string line, ILoggerFlags flags)
        {
            WriteLine(line, flags, String.Empty);
        }
        public void WriteLine(string line, ILoggerFlags flags, string source)
        {
            if ((this._flags & flags) == flags)
            {
                if (string.IsNullOrEmpty(source))
                {
                    try
                    {
                        StackTrace stackTrace = new StackTrace();
                        source = stackTrace.GetFrame(1).GetFileName() + @"\" +
stackTrace.GetFrame(1).GetMethod().Name;
                    }
                    catch
                    {
                        source = null;
                    }
                }
                Console.WriteLine("[" +
(!string.IsNullOrEmpty(Thread.CurrentThread.Name) ? (Thread.CurrentThread.Name + "/")
: "") + Thread.CurrentThread.ManagedThreadId + "+" + source + "] " + line);
            }
        }
```

```
        #endregion
    }
}
```

2.1.6 Loggers\WSLogger.cs

ILogger implementation that Uses FGBWeb_General Web Service to log to a table in the FGBFCS database.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using System.Threading;
using FGBFaultDetectionServiceKernel.FGBWeb_General;
namespace FGBFaultDetectionServiceKernel.Loggers
{
    /// <summary>
    /// Uses FGBWeb_General Web Service to log to a table in the database.
    /// </summary>
    class WSLogger : ILogger
    {
        General _ws;
        ILoggerFlags _flags = (ILoggerFlags.NORMAL | ILoggerFlags.ERRORS);

        public string Name = "WebServiceLogger";

        public WSLogger(General ws)
        {
            this._ws = ws;
        }
        public WSLogger(General ws, ILoggerFlags flags)
        {
            this._ws = ws;
            this.setFlags(flags);
        }

        #region ILogger Members
        public string GetName()
        {
            return this.Name;
        }

        public void WriteLine(string line)
        {
            this.WriteLine(line, ILoggerFlags.NORMAL);
        }

        public void setFlags(ILoggerFlags flags)
        {
            this._flags = flags;
        }
        public void WriteLine(string line, ILoggerFlags flags)
        {
            this.WriteLine(line, flags, string.Empty);
        }
        /// <summary>
        /// Inserts a new record to the FaultDetection_Log table.
        /// </summary>
        /// <param name="line">the line to insert</param>
        /// <param name="flags">the flags of the line to insert</param>
```

```
        /// <param name="source">the source of the logging, if null or empty, will be
automatically detected.</param>
        public void WriteLine(string line, ILoggerFlags flags, string source)
        {
            if (_ws != null && ((this._flags & flags) > 0))
            {
                try
                {
                    if (string.IsNullOrEmpty(source))
                    {
                        try
                        {
                            // find out which filename/method called this method.
                            StackTrace stackTrace = new StackTrace();
                            source = stackTrace.GetFrame(1).GetFileName() + @"\" +
stackTrace.GetFrame(1).GetMethod().Name;
                        }
                        catch
                        {
                            source = null;
                        }
                    }
                    // prepend the current thread id to the log line and save it to
the database.
                    _ws.FaultDetectionLogInsert("[" +
(!string.IsNullOrEmpty(Thread.CurrentThread.Name) ? (Thread.CurrentThread.Name + "/")
: "") + Thread.CurrentThread.ManagedThreadId + "] " + line, (int)flags, source);
                }
                catch
                {

                }
            }
        }

        #endregion
    }

}
```

2.2 Fault Detection Monitoring Service

In the service itself, only 2 classes were implemented. The FGBFaultDetectionService which is the main class that loads the Kernel and starts it and an ***ILogger*** implementation that logs to Windows Event Viewer.

2.2.1 FGBFaultDetectionService.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.ServiceProcess;
using System.Text;
using FGBFaultDetectionServiceKernel;
using FGBFaultDetectionService.Classes;
namespace FGBFaultDetectionService
{
    public partial class FGBFaultDetectionService : ServiceBase
```

```
    {
        new public const String ServiceName = "FGBFaultDetectionService";
        Kernel kernel;
        ILogger eventViewerLogger;

        public FGBFaultDetectionService()
        {
            InitializeComponent();
            eventViewerLogger = new EventViewerLogger(ServiceName);
            try
            {
                kernel = new Kernel();
                kernel.AddLogger(eventViewerLogger);
            }
            catch (Exception ex)
            {
                if (eventViewerLogger != null)
                {
                    eventViewerLogger.WriteLine("Shutting Down -> Failed to initalize
Kernel: " + ex.Message, ILoggerFlags.ERRORS);
                }
                this.Stop();
            }
        }

        /// <summary>
        /// Overrides the base OnStart method.
        ///
        /// Starts the underlying FaultDetectionServiceKernel object.
        /// </summary>
        /// <param name="args"></param>
        protected override void OnStart(string[] args)
        {
            try
            {
                kernel.Start();
            }
            catch (Exception ex)
            {
                if (eventViewerLogger != null)
                {
                    eventViewerLogger.WriteLine("Shutting Down -> Failed starting
Kernel: " + ex.Message, ILoggerFlags.ERRORS);
                }
                this.Stop();
            }
        }
        /// <summary>
        /// Overrides the base OnStop method.
        ///
        /// Stops the underlying FaultDetectionServiceKernel object.
        /// </summary>
        protected override void OnStop()
        {
            kernel.Stop("Service.OnStop Called");
        }
        /// <summary>
        /// Overrides the base OnShutdown method.
        ///
        /// Stops the underlying FaultDetectionServiceKernel object.
        /// </summary>
        protected override void OnShutdown()
```

```
        {
            kernel.Stop("Service.OnShutdown Called");
        }

        private void log(string line)
        {
            if (this.eventViewerLogger != null)
            {
                this.eventViewerLogger.WriteLine(line);
            }
        }
    }
}
```

2.2.2 EventViewerLogger.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using FGBFaultDetectionServiceKernel;
using System.Diagnostics;

namespace FGBFaultDetectionService.Classes
{
    /// <summary>
    /// An ILogger implementation which logs to Windows Event Viewer.
    /// </summary>
    class EventViewerLogger : FGBFaultDetectionServiceKernel.ILogger
    {
        private const string _name = "EventViewerLogger";
        private string _serviceName = "FGBFaultDetectionService";
        private ILoggerFlags _flags = ILoggerFlags.EVENTLOG;

        public static EventLog eventLog = new EventLog();

        public EventViewerLogger(string serviceName)
        {
            this._serviceName = serviceName;
            LoadEventLog(this._serviceName);
        }

        #region ILogger Members

        public string GetName()
        {
            return _name;
        }
        public void setFlags(ILoggerFlags flags)
        {
            this._flags = flags;
        }
        /// <summary>
        /// Writes a new entry to Windows Event Log
        /// </summary>
        /// <param name="line">the line/content to write</param>
        public void WriteLine(string line)
        {
            WriteLine(line, ILoggerFlags.NORMAL);
        }
        /// <summary>
        /// Writes a new entry to Windows Event Log
        /// </summary>
```

```
        /// <param name="line">the line/content to write</param>
        /// <param name="flags">the ILogger flags for this line</param>
        public void WriteLine(string line, ILoggerFlags flags)
        {
            this.WriteLine(line, flags, String.Empty);
        }
        /// <summary>
        /// Writes a new entry to Windows Event Log
        /// </summary>
        /// <param name="line">the line/content to write</param>
        /// <param name="flags">the ILogger flags for this line</param>
        /// <param name="source">where the logging is happening</param>
        public void WriteLine(string line, ILoggerFlags flags, string source)
        {
            if ((this._flags & flags) == flags)
            {
                if (eventLog != null)
                {
                    try
                    {
                        eventLog.WriteEntry(line);
                    }
                    catch //(Exception ex)
                    {

                    }
                }
            }
        }

        #endregion

        private void LoadEventLog(string serviceName)
        {
            try
            {
                if (!System.Diagnostics.EventLog.SourceExists(serviceName))
                {
                    System.Diagnostics.EventLog.CreateEventSource(serviceName,
"Application");
                }
                eventLog.Source = serviceName;
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex.StackTrace);
            }
        }
    }
}
```

3. Fault Detection Web API and Web-Based Explanation Facility

The Web API and Web-Based Explanation Facility were built in the same "ASP.NET MVC 4 Web Application" project. In this type of Visual Studio project, the API controllers and HTML views (i.e. web pages) are separated.

3.1 App_Start

The App_Start folder is auto-generated and holds mainly auto generated classes, however, for this project a couple of these had to be edited.

3.1.1 JsonContentNegotiator.cs

Using a custom Content Negotiatior to make sure all communication with the API is done in JSON.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Net.Http.Formatting;
using System.Net.Http;
using System.Net.Http.Headers;

namespace FGBFaultDetectionWebApplication
{
    /// <summary>
    /// Using a custom Content Negotiatior to make sure all communication with the API
is done in JSON.
    /// </summary>
    public class JsonContentNegotiator : IContentNegotiator
    {
        private readonly JsonMediaTypeFormatter _jsonFormatter;

        public JsonContentNegotiator(JsonMediaTypeFormatter formatter)
        {
            _jsonFormatter = formatter;
        }

        public ContentNegotiationResult Negotiate(Type type, HttpRequestMessage
request, IEnumerable<MediaTypeFormatter> formatters)
        {
            var result = new ContentNegotiationResult(_jsonFormatter, new
MediaTypeHeaderValue("application/json"));
            return result;
        }
    }
}
```

3.1.2 WebApiConfig.cs

Auto-generated class that configures the different Web API Controllers. The manual changes done are commented inline.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using System.Net.Http.Formatting;
using Newtonsoft.Json.Serialization;

namespace FGBFaultDetectionWebApplication
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{identifier}/{failedOnly}",
                defaults: new { identifier = RouteParameter.Optional, failedOnly =
RouteParameter.Optional }
            );

            // setting a new formatter, so JSON will be serialised without "Field"
being appended to every object/class property.
            var jsonFormatter = new JsonMediaTypeFormatter();
            // preventing the contract resolver from adding "Field" to the end of
attribute/properties which are not marked as [DataMember]
            DefaultContractResolver camelCase = new DefaultContractResolver();
            jsonFormatter.SerializerSettings.ContractResolver = camelCase; // new
CamelCasePropertyNamesContractResolver();
            config.Formatters.JsonFormatter.SerializerSettings.ContractResolver =
camelCase;// new CamelCasePropertyNamesContractResolver();

            // replacing the default content negotiator with a JSON one, so all
communication with the API will be done in JSON.
            config.Services.Replace(typeof(IContentNegotiator), new
JsonContentNegotiator(jsonFormatter));
        }
    }
}
```

3.2 Models

There were no specific models defined in this project. All models are based on Fortress Web Service objects (i.e. TMCTillInfoFaultDetection) and the FDI System objects (i.e. FaultDetectionExecution and).

3.3 Views

3.3.1 Views/Home/Index.cshtml

The only view available. This is the Explanation Facility Web Page which consumes the Web API. It is built using knockout.js (www. knockoutjs.com) and JQuery (www.jquery.com), two widely used and tests JavaScript Libraries.

```
<div id="loading" class="modal">
    <div id="loading-holder" class="loading-holder">
        <div class="loading-caption">loading...</div>
        <div class="loading-icon"></div>
        <div class="clear"></div>
    </div>
</div>
<header>
    <div class="content-wrapper">
        <div class="float-left">
            <p class="site-title">
                <a href="~/">TMC FaultDetection Monitor</a></p>
        </div>
    </div>
</header>
<div id="body">
    <section class="content-wrapper main-content clear-fix">

            <div class="toolbar">
                <div class="search">
                    <span><input type="text" data-bind="value: searchString"
placeholder="Search" /></span>
                    <div class="icon" data-bind="click: getTMCs"></div>
                    <div class="clear"></div>
                </div>

                <div data-bind="attr: { class: 'button check'+(displayOnlyFaulty() ? '
checked' : '') },  click: toggleDisplay">
                    <span class="icon"></span>
                    <span data-bind="text: (displayOnlyFaulty() ? 'Displaying Faulty
Only' : 'Display Faulty Only')"></span>
                </div>

                <div class="clear"></div>
            </div>
            <div data-bind="if: tmcsLoaded()">
                <div id="tmclist" data-bind="foreach: tmcs()">
                        <div data-bind="attr: { class: 'tmc '+ ($index() % 2 == 0 ?
'': 'odd') + (FaultDetectionResult() > 0 ? ' error' : (FaultDetectionResult() < 0 ? '
notchecked' : '')) + ((!$root.displayOnlyFaulty() || FaultDetectionResult() > 0) ? ''
: ' hidden') }, if: (!$root.displayOnlyFaulty() || FaultDetectionResult() != 0)">
                            <div class="cell name">
                                <div class="main" data-bind="text: TillName()"></div>
                                <div class="sub"><span>POS-ID: </span><span data-
bind="text: PosID()"></span></div>
                            </div>
                            <div class="cell addresses">
```

```
                                <div class="main"><span>IP: </span><span data-
bind="text: IP()"></span></div>
                                <div class="sub"><span>Mac-Addres: </span><span data-
bind="text: MacAddress()"></span></div>
                            </div>
                            <div class="cell information">
                                <div class="main"><span>Last Online: </span><span
data-bind="text: LastOnline()"></span></div>
                                <div class="sub"><span>Version: </span><span data-
bind="text: TMCVersion()"></span></div>
                            </div>
                            <div class="cell faultdetection">
                                <div class="icon-holder">
                                    <div class="icon" data-bind="if:
(FaultDetectionResult() == 1), click: $root.openFaultDetection"></div>
                                    <div class="details">
                                        <span data-bind="text: (FaultDetectionResult()
!= 0 && FaultDetectionError().length > 0 ? FaultDetectionError() : 'All
Good')"></span></div>

                                    <div class="clear"></div>
                                </div>
                                <div class="sub">
                                    <span data-bind="text:
FaultDetectionLastUpdated()"></span>
                                </div>
                            </div>

                            <div class="clear"></div>
                        </div>
                    </div>
                </div>
                <div data-bind="ifnot: (tmcsLoaded())">
                No TMCs Found to Display.
                </div>
    </section>


</div>
<div id="faultDetectionDialog">
    <div data-bind="if: (faultDetectionExecutionError().length == 0)">
        <div data-bind="with: faultDetectionExecution">
            <h3>
                <span>Execution ID: </span>
                <span data-bind="text: ID()"></span> (<span data-bind="text:
Time()"></span>)
                <span class="button" data-bind="click: $root.recheck">Recheck</span>
            </h3>
            <div data-bind="with: FaultCollection">
                <div data-bind="with: Module">
                    <h2>
                        <span data-bind="text: TypeName()"></span>
                        <span data-bind="text: UniqueName"></span>
                        <span>(<span data-bind="text: UniqueId"></span>)</span>
                    </h2>
                    <ul data-bind="foreach: Properties">
                        <li><span data-bind="text: Name()"></span>: <span data-
bind="text: Value()"></span></li>
                    </ul>
                </div>
                <div data-bind="foreach: List">
                    <div class="fault-detection-case">
```

```
                        <div class="fault-detection-case-name">
                            <span class="name" data-bind="text: Name()"></span>
                            <span class="error-holder" data-bind="if:
(Result().Error().length > 0)">
                                (<span class="error-text" data-bind="text:
Result().Error()"></span>)
                            </span>
                        </div>

                        <div data-bind="foreach: Tests">
                            <div class="fault-detection-test">
                                <div class="fault-detection-test-name">
                                    <span class="name" data-bind="text:
Name()"></span>
                                    <span class="error-holder" data-bind="if:
(Result().Error().length > 0)">
                                        (<span class="error-text" data-bind="text:
Result().Error()"></span>)
                                    </span>
                                </div>
                                <div class="fault-detection-test-params-holder">
                                    <table class="fault-detection-test-params-table"
data-bind="foreach: Rules">
                                        <tr data-bind="attr: { class:
(Result().Passed() ? 'test-params-passed' : 'test-params-failed') + ((Group() % 2 ==
0) ? ' test-params-group ' : ' test-params-group odd') }">
                                            <td>
                                                <div class="fault-detection-test-
param">
                                                    <div class="" data-bind="attr:{
class: 'fault-detection-test-param-value '+Param1().cssType() }, text:
Param1().Value()"></div>
                                                    <div data-bind="if:
(Param1().Value() != Param1().ComputedValue())">
                                                        <div class="" data-bind="attr:
{ class: 'fault-detection-test-param-computedvalue '+Param1().cssComputedType() },
text: Param1().ComputedValue()"></div>
                                                        <div class="clear"></div>
                                                    </div>
                                                    <div class="fault-detection-test-
param-details">
                                                        <span class="fault-detection-
test-param-type" data-bind="text: Param1().Type()"></span>
                                                        <span class="" data-
bind="attr: { class: 'fault-detection-test-param-computedtype' + (Param1().Type() ==
Param1().ComputedType() ? ' hidden' : '') }, text: Param1().ComputedType()"></span>

                                                    </div>
                                                </div>
                                            </td>
                                            <td class="fault-detect-test-param-
operation" data-bind="text: ((Negate() ? 'NOT ' : '') + OperationTypeName())"></td>
                                            <td>
                                                <div class="fault-detection-test-
param">
                                                    <div class="" data-bind="attr:{
class: 'fault-detection-test-param-value '+Param2().cssType() }, text:
Param2().Value()"></div>
                                                    <div data-bind="if:
(Param2().Value() != Param2().ComputedValue())">
                                                        <div class="" data-bind="attr:
{ class: 'fault-detection-test-param-computedvalue '+Param2().cssComputedType() },
```

```
text: Param2().ComputedValue()"></div>
                                                        <div class="clear"></div>
                                                    </div>
                                                    <div class="fault-detection-test-
param-details">
                                                        <span class="fault-detection-
test-param-type" data-bind="text: Param2().Type()"></span>
                                                        <span class="" data-
bind="attr: { class: 'fault-detection-test-param-computedtype' + (Param2().Type() ==
Param2().ComputedType() ? ' hidden' : '') }, text: Param2().ComputedType()"></span>
                                                    </div>
                                                </div>
                                            </td>
                                            <td data-bind="if:
(Result().Error().length > 0)">
                                                <div class="fault-detection-test-
param-error error" data-bind="text: Result().Error()">

                                                </div>
                                            </td>
                                        </tr>
                                        <tr data-bind="if: (RelationType() !=
'NONE')">
                                            <td colspan="3" class="fault-detect-test-
param-relation" data-bind="text: RelationType()"></td>
                                        </tr>
                                    </table>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div data-bind="if: (faultDetectionExecutionError().length > 0)">
        <div class="error" data-bind="text: faultDetectionExecutionError()"></div>
    </div>
</div>
<script>
    // a JavaScript/Knockout.js version of TMCTillInfoFaultDetection object.
    var tmc = function (item) {
        var self = this;

        self.MacAddress = ko.observable(item.MacAddress);
        self.PosID = ko.observable(item.PosID);
        self.IP = ko.observable(item.IP);
        self.TillName = ko.observable(item.TillName);
        self.TMCVersion = ko.observable(item.TMCVersion);
        self.GPSOCXVersion = ko.observable(item.GPSOCXVersion);
        self.IPCVersion = ko.observable(item.IPCVersion);
        self.PrintServerVersion = ko.observable(item.PrintServerVersion);
        self.IsCashBackEnabled = ko.observable(item.IsCashBackEnabled);
        self.IsCashBackOnCashBackEnabled =
ko.observable(item.IsCashBackOnCashBackEnabled);
        self.DELETED = ko.observable(item.DELETED);
        self.ConfomationLimit = ko.observable(item.ConfomationLimit);
        self.LastOnline = ko.observable(item.LastOnline);

        self.FaultDetectionResult = ko.observable(item.FaultDetectionResult);
        self.FaultDetectionExecutionId =
ko.observable(item.FaultDetectionExecutionId);
```

```
        self.FaultDetectionError = ko.observable(item.FaultDetectionError);
        self.FaultDetectionLastUpdated =
ko.observable(item.FaultDetectionLastUpdated);

        self.genCSS = function (index) {
            return ((index % 2 <= 0 ? '' : 'odd')) + (FaultDetectionResult() > 0 ? '
error' : '');
        }

    }
    // a JavaScript/Knockout.js version of FaultDetectionExecution object
    var faultDetectionExecution = function (item) {
        var self = this;

        self.ID = ko.observable(item.ID);
        self.Deleted = ko.observable(item.Deleted);
        self.Time = ko.observable(item.Time);
        self.Duration = ko.observable(item.Duration);

        self.FaultCollection = ko.observable(new
faultDetectionCollection(item.FaultCollection));

        self.loadCollection = function (collection) {
            self.FaultCollection().loadCases(collection.List);
        }
    }
    // a JavaScript/Knockout.js version of FaultDetectionCaseCollection object
    var faultDetectionCollection = function (item) {
        var self = this;

        self.List = ko.observableArray([]);
        self.Module = ko.observable(new faultDetectionModule(item.Module));
        self.Result - ko.observable(new faultDetectionResult(item.Result));

        self.loadCases = function (list) {
            for (var i in list) {
                var fdCase = new faultDetectionCase(list[i]);
                fdCase.loadTests(list[i]);
                fdCase.loadNotificationTeams(list[i]);
                self.List.push(fdCase);
            }
        }
    }
    // a JavaScript/Knockout.js version of FaultDetectionCase object
    var faultDetectionCase = function (item) {
        var self = this;

        self.ID = ko.observable(item.ID);
        self.ModuleTypeID = ko.observable(item.ModuleTypeID);
        self.Name = ko.observable(item.Name);
        self.Description = ko.observable(item.Description);
        self.ModuleTypeName = ko.observable(item.ModuleTypeName);
        self.ModuleName = ko.observable(item.ModuleName);
        self.Tests = ko.observableArray([]);
        self.NotificationTeams = ko.observableArray([]);
        self.Result = ko.observable(new faultDetectionResult(item.Result));

        self.loadTests = function (collection) {
            for (var i in collection.Tests) {
                var fdTest = new faultDetectionTest(collection.Tests[i]);
                fdTest.loadRules(collection.Tests[i]);
                self.Tests.push(fdTest);
```

```
            }
        }
        self.loadNotificationTeams = function (collection) {
            for (var i in collection.NotificationTeams) {
                var fdTeam = new faultDetectionTeam(collection.NotificationTeams[i]);
                self.NotificationTeams.push(fdTeam);
            }
        }

    }
    // a JavaScript/Knockout.js version of FaultDetectionTeam object
    var faultDetectionTeam = function (item) {
        var self = this;

        self.ID = ko.observable(item.ID);
        self.Name = ko.observable(item.Name);
        self.Email = ko.observable(item.Email);
        self.Flags = ko.observable(item.Flags);

    }
    // a JavaScript/Knockout.js version of FaultDetectionTest object
    var faultDetectionTest = function (item) {
        var self = this;

        self.ID = ko.observable(item.ID);
        self.CaseID = ko.observable(item.ID);
        self.Name = ko.observable(item.Name);
        self.SuccessMessage = ko.observable(item.IDSuccessMessage);
        self.FailMessage = ko.observable(item.FailMessage);
        self.Rules = ko.observableArray([]);
        self.Result = ko.observable(new faultDetectionResult(item.Result));
        self.Groups = ko.observable(item.Groups);

        self.loadRules = function (test) {
            for (var i in test.Rules) {
                var fdTestRule = new faultDetectionTestRule(test.Rules[i]);
                self.Rules.push(fdTestRule);
            }
        }
    }
    // a JavaScript/Knockout.js version of FaultDetectionTestRule object
    var faultDetectionTestRule = function(item) {
        var self = this;

        self.Param1 = ko.observable(new faultDetectionParam(item.Param1));
        self.Param2 = ko.observable(new faultDetectionParam(item.Param2));
        self.TestId = ko.observable(item.TestId);
        self.Id = ko.observable(item.Id);
        self.Group = ko.observable(item.Group);
        self.Order = ko.observable(item.Order);
        self.RelationType = ko.observable(item.RelationType);
        self.OperationTypeName = ko.observable(item.OperationTypeName);
        self.Negate = ko.observable(item.Negate);
        self.Result = ko.observable(new faultDetectionResult(item.Result));
        self.FaultDetectionRelationType =
ko.observable(item.FaultDetectionRelationType);
    }
    // a JavaScript/Knockout.js version of FaultDetectionParam object
    var faultDetectionParam = function (item) {
        var self = this;

        self.Type = ko.observable(item.Type);
```

```
        self.Value = ko.observable(item.Value);
        self.ID = ko.observable(item.ID);
        self.ComputedType = ko.observable(item.ComputedType);
        self.ComputedValue = ko.observable((item.ComputedType == "NULL" &&
item.ComputedValue.length == 0) ? 'null' : item.ComputedValue);
        self.FaultDetectionParamType = ko.observable(item.FaultDetectionParamType);

        self.cssType = ko.computed(function () {
            //console.log(self.ID()+' '+self.Value()+' - type-' + self.Type());
            //console.log('type-' + self.Type().toLowerCase());
            //console.log(self.Type().toLowerCase());
            //console.log(self.Type());
            return "type_" + (self.Type() != null && self.Type() instanceof String ?
self.Type().toLowerCase() : 'null');
        });
        self.cssComputedType = ko.computed(function () {
            console.log(self.ComputedType());
            return "type_" + (self.ComputedType() != null && self.ComputedType()
instanceof String ? self.ComputedType().toLowerCase() : 'null');
        });
    }
    // a JavaScript/Knockout.js version of IFaultDetectionModule object
    var faultDetectionModule = function (item) {
        var self = this;

        self.TypeID = ko.observable(item.TypeID);
        self.TypeName = ko.observable(item.TypeName);
        self.UniqueId = ko.observable(item.UniqueId);
        self.UniqueName = ko.observable(item.UniqueName);
        self.Properties = ko.observableArray([]);

        self.loadProperties = function (properties) {
            for (var i in properties) {
                //console.log(i);
                //console.log(properties[i]);
                self.Properties.push(new faultDetectionModuleProperty({ 'Name': i,
'Value': properties[i] }));
            }
        }
    }
    // a JavaScript/Knockout.js version of IFaultDetectionModule property
    var faultDetectionModuleProperty = function (item) {
        var self = this;

        self.Name = ko.observable(item.Name);
        self.Value = ko.observable(item.Value);
    }
    // a JavaScript/Knockout.js version of FaultDetectionResult object
    var faultDetectionResult = function (item) {
        var self = this;
        //console.log(item);
        self.Executed = ko.observable(item.Executed);
        self.Passed = ko.observable(item.Passed);
        self.Error = ko.observable(item.Error != null ? item.Error : '');
    }
    // A Knockout.js ViewModel used to control the views and models of the explanation
faciltiy web page
    var viewModel = function () {
        var self = this;
        self.searchString = ko.observable("");
        self.totalRecords = ko.observable(0);
        self.displayOnlyFaulty = ko.observable(true);
```

```
        self.tmcsLoaded = ko.observable(false);

        self.tmcs = ko.observableArray([]);
        self.faultDetectionExecution = ko.observable(null);
        self.faultDetectionExecutionMacAddress = ko.observable('');
        self.faultDetectionExecutionError = ko.observable("");
        self.faultDetectionTMC = ko.observable(null);

        // get a list of tmcs by calling /api/tmc or /api/tmc/?s=query when a search
is done.
        self.getTMCs = function (dateString, ignoreAlreadyPrintedChildren,
currentPageIndex) {
            //console.log('getGuardiansByDate: ' + dateString);
            self.tmcsLoaded(false);
            $.ajax({
                type: "get",
                url: "api/tmc/" + (self.searchString().length > 0 ? "?s=" +
self.searchString() : ""),
                data: '',
                dataType: "json",
                contentType: "application/json; charset=utf-8",
                success: function (result) {
                    if (result != null && result.length > 0) {
                        self.loadTMCs(result);
                    }
                    else {
                        console.log("getTMCs.invalid_result: " + result);
                    }
                },
                failure: function (result) {
                    console.log("getTMCs.fail: " + result);

                }
            });
        }
        // clears out the current list of tmcs and loads a new list based on the
results of getTMCs
        self.loadTMCs = function (data) {
            self.tmcs.removeAll();
            //console.log(data);
            for (var i in data) {
                self.tmcs.push(new tmc(data[i]));
            }
            self.tmcsLoaded(self.tmcs().length > 0);
            //self.loadGuardiansCurrentPage();
        }
        // toggles the display (i.e. display tmcs which had faults detected on, or all
tmcs)
        self.toggleDisplay = function () {
            self.displayOnlyFaulty(!self.displayOnlyFaulty());
        }
        // loads the past fault detection execution dialog to allow end-users to see
why/how a fault was detected.
        self.openFaultDetection = function (item) {
            //console.log(item);
            //console.log(item.FaultDetectionExecutionId());
            if (item.FaultDetectionResult() == 1) {
                self.faultDetectionExecutionMacAddress(item.MacAddress());
                self.getFaultDetectionExecution(item);
                faultDetectionDialog.dialog("open");
            }
        }
```

```
        // loads a fault detection execution (used in the dialog)
        self.loadFaultDetectionExecution = function (item) {
            self.faultDetectionExecution(new faultDetectionExecution(item));
            //console.log(item.FaultCollection);
            self.faultDetectionExecution().loadCollection(item.FaultCollection);

self.faultDetectionExecution().FaultCollection().Module().loadProperties(item.FaultCol
lection.Module.Properties);

            if (self.faultDetectionTMC() != null &&
self.faultDetectionTMC().FaultDetectionExecutionId() != item.ID) {
                //console.log(item);
                var executionResult = (item.FaultCollection != null &&
item.FaultCollection.Result != null ? (item.FaultCollection.Result.Passed ? 0 : 1) : -
1);
                //console.log(executionResult);
                self.faultDetectionTMC().FaultDetectionResult(executionResult);
                self.faultDetectionTMC().FaultDetectionExecutionId(item.ID);
                self.faultDetectionTMC().FaultDetectionLastUpdated(item.Time);
                self.faultDetectionTMC().FaultDetectionError('[RECHECK] ' +
item.FaultCollection.Result.Error);
            }

        }
        // gets a past fault detection execution by its id using
/api/faultdetection/id
        self.getFaultDetectionExecution = function (item) {
            //console.log(item.FaultDetectionExecutionId());
            self.faultDetectionExecutionError('');
            self.faultDetectionTMC(item);
            $.ajax({
                type: "get",
                url: 'api/faultdetection/' + item.FaultDetectionExecutionId() + '/1',
                data: '',
                dataType: "json",
                contentType: "application/json; charset=utf-8",
                success: function (result) {
                    if (result != null && result.ID > 0 && result.FaultCollection !=
null) {
                        self.loadFaultDetectionExecution(result);
                    }
                    else {
                        self.faultDetectionExecutionError('Failed getting Fault
Detection Execution (id: ' + item.FaultDetectionExecutionId() + ')');
                    }
                },
                failure: function (result) {
                    console.log("getFaultDetectionExecution.fail: " + result);
                    self.faultDetectionExecutionError('Unknown error happend when
trying to get Fault Detection Execution (id: ' + item.FaultDetectionExecutionId() +
'), Please try again later.');
                }
            });
        }
        // executes a recheck on the currently open tmc (i.e. fault detection
execution dialog open) by posting its mac address to /api/faultdetection/
        self.recheck = function () {
            if (self.faultDetectionExecutionMacAddress() != null &&
self.faultDetectionExecutionMacAddress().length > 0) {
                $.ajax({
                    type: "post",
                    url: 'api/faultdetection/',
```

```
                    data: '\'' + self.faultDetectionExecutionMacAddress() + '\'',
                    dataType: "json",
                    contentType: "application/json; charset=utf-8",
                    success: function (result) {
                        if (result != null && result.ID > 0 && result.FaultCollection
!= null) {

                            self.loadFaultDetectionExecution(result);
                        }
                        else {
                            self.faultDetectionExecutionError('Failed getting Recheck
Result (' + result + ')' + (result == null ? ' Please verify that this TMC is connect
to the network' : ''));
                        }
                    },
                    failure: function (result) {
                        console.log("recheck.fail: " + result);
                        self.faultDetectionExecutionError('Unknown Error while getting
Recheck Result (' + result + ')');
                    },
                    error: function (jqXHR, textStatus, errorThrown) {
                        console.log("recheck.fail: " + jqXHR);
                        console.log("recheck.fail: " + textStatus);
                        console.log("recheck.fail: " + errorThrown);
                        self.faultDetectionExecutionError(jqXHR.responseText);
                    }
                });
            }
        }
    }
    // shows the loading panel
    function showLoading(show) {
        if (show) {
            $('#loading').fadeIn(100);
            $('#loading-holder').slideDown(300);
        }
        else {
            $('#loading').hide();
        }
    }

    var viewModelHolder = new viewModel();
    var faultDetectionDialog;

    // use jquery to bind knockout.js, create the fault detection execution dialog and
load an initial list of tmcs.
    $(document).ready(function () {
        ko.applyBindings(viewModelHolder);
        faultDetectionDialog = $('#faultDetectionDialog').dialog({ title: 'Fault
Detection Result', autoOpen: false, height: 600, width: 940 });
        viewModelHolder.getTMCs();
    });
    // set a global jquery ajax start callback function to show the loading panel
every time an ajax request is being executed.
    $(document).ajaxStart(function () {
        showLoading(true);
    });
    // set a global jquery ajax complete callback function to hide the loading panel
every time an ajax request has completed.
    $(document).ajaxComplete(function () {
        showLoading(false);
    });
```

```
</script>
```

3.4 Controllers

There are two types of Controllers in this type of project. A general Controller used for “Views” and an API Controller used to get get data to populate views.

3.4.1 FaultDetectionController.cs

An API Controller used to get past Fault Detection Executions and to execute a Recheck on a TMC based on its MAC address.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using FaultDetection;
using System.Configuration;
using FaultDetection.Environments;
using System.Text.RegularExpressions;
using FaultDetection.Modules;
using FGBFaultDetectionWebApplication.FGBWeb_TMC;
using FGBFaultDetectionWebApplication.FGBWeb_TMCWeb;
namespace FGBFaultDetectionWebApplication.Controllers
{
    /// <summary>
    /// Fault Detection Controller
    /// Used to get past Fault Detection Executions and to execute a Recheck on a TMC
based on its MAC address.
    /// </summary>
    public class FaultDetectionController : ApiController
    {

        // GET /api/faultdetection/id
        /// <summary>
        /// Gets a complete past fault detection execution based on given Id
        /// </summary>
        /// <param name="id">the id of the past fault detection execution to
get</param>
        /// <returns>a complete FaultDetectionExecution representing a past fault
detection execution check</returns>
        public FaultDetectionExecution Get(int identifier)
        {
            return Get(identifier, 0);
        }
        // GET /api/faultdetection/id/1
        /// <summary>
        /// Gets a complete past fault detection execution based on given Id
        /// Enables getting only the failed fault cases (i.e. only detected faults) by
using 1 for failedOnly.
        /// </summary>
        /// <param name="id">the id of the past fault detection execution to
get</param>
        /// <param name="failedOnly">whether or not to only get detected faults (1) or
get all fault cases from the execution (0)</param>
        /// <returns>a complete FaultDetectionExecution representing a past fault
detection execution check</returns>
        public FaultDetectionExecution Get(int identifier, int failedOnly)
        {
```

```
            IFaultDetectionKnowledgebase db =
FaultDetection.FaultDetectionKnowledgebaseFactory.Create(ConfigurationManager.Connecti
onStrings["xFGBFaultDetection"].ConnectionString);
            if (db != null) {
                FaultDetectionExecution execution =
db.getFaultDetectionExecution(identifier, (failedOnly == 1));
                return execution;
            }
            return null;
        }
        // POST /api/faultdetection/ 'macaddress'
        /// <summary>
        /// Executes a "recheck" - a new fault detection check on a specific TMC (by
its Mac Address).
        ///
        /// Gets a TMCTillInfoFaultDetection object from FGBWeb_TMC Web Service, gets
its details and IP
        /// Uses FGBWeb_TMCWeb to get the state of that TMC
        /// Uses the Fault Detection DLL to execute a Fault Detection Tests. Logs and
return the results.
        /// </summary>
        /// <param name="macaddress">the mac address of the TMC to recheck</param>
        /// <returns>FaultDetectionExecution representing the full results of the
check (i.e. both detected faults and fault cases which were okay)</returns>
        public FaultDetectionExecution Post([FromBody] string macaddress)
        {
            FaultDetectionExecution execution = null;
            try
            {
                TMC wsTMC = new FGBWeb_TMC.TMC();
                wsTMC.Url = ConfigurationManager.AppSettings.Get("FGB_WebServiceURL")
+ "tmc.asmx";
                TMCTillInfoFaultDetection tmc = wsTMC.TMCGetByMacAddress(macaddress);
                if (tmc != null)
                {

                    FaultDetectionEnvironmentFGBFCS env = loadFGBFCSEnvironment();
                    FaultDetectionModuleTMCX moduleTMCX =
loadTMCModuleFromTMCTillInfoFaultDetection(tmc);
                    TMCResponse tmcResponse = TMCWSCallSimple(tmc.IP);
                    if (tmcResponse != null && tmcResponse.callResult.ErrorNumber ==
0)
                    {
                        string tmcStatus =
System.Text.Encoding.Default.GetString(FaultDetection.FaultDetectionUtil.DecodeHexStri
ng(tmcResponse.Message.SpareStr1));
                        int loaded = moduleTMCX.loadStatus(tmcStatus);
                        if (loaded >= 0)
                        {
                            string conString =
ConfigurationManager.ConnectionStrings["xFGBFaultDetection"].ConnectionString;
                            IFaultDetectionKnowledgebase kb =
FaultDetection.FaultDetectionKnowledgebaseFactory.Create(conString);
                            if (kb != null)
                            {
                                FaultDetectionCaseCollection fdCollection =
kb.getFaultDetectionCollection(moduleTMCX);
                                if (fdCollection != null && fdCollection.List.Count >
0)
                                {
                                    FaultDetectionEngine engine = new
FaultDetectionEngine(env);
```

```
                                    FaultDetectionResult result = engine.execute(ref
fdCollection);
                                    execution =
kb.saveFaultDetectionExecution(fdCollection);
                                    saveTMCExecution(tmc, result, execution);
                                }
                                else
                                {
                                    throw new Exception("Failed getting Fault
Detection Case Collection from Knowledgebase (" + conString + ")");
                                }
                            }
                            else
                            {
                                throw new Exception("Failed creating FaultDetection DB
Object, check connection string (" + conString + ")");
                            }
                        }
                    }
                    else
                    {
                        throw new Exception("Getting TMC Status Failed" + tmcResponse
!= null ? " " + tmcResponse.Message.Msg1 + " / " +
tmcResponse.callResult.ErrorDescription + " (" + tmcResponse.callResult.ErrorNumber +
")" : "");
                    }
                }
                else
                {
                    throw new Exception("Failed Getting TMC details from DB, please
check the Mac Address exists in TMC_TillInfo table");
                }
            }
            // catching any web service exception and returning it a
HttpResponseException to establish uniform replies (parsed by knockout.js on the
explanation facility web page)
            catch (WebException wex)
            {
                throw new
HttpResponseException(Request.CreateResponse((HttpStatusCode)502, "WebService Error:
"+ wex.Message));
            }
            // catching any exception and returning it a HttpResponseException to
establish uniform replies (parsed by knockout.js on the explanation facility web page)
            catch (Exception ex)
            {
                throw new
HttpResponseException(Request.CreateResponse((HttpStatusCode)501, ex.Message));
            }
            return execution;
        }
        /// <summary>
        /// Loads an FGBFCS Envrionment Object using FGBWeb_General Web Service
        /// </summary>
        /// <returns>returns a FaultDetectionEnvironmentFGBFCS representing the
current FGBFCS environment</returns>
        private FaultDetectionEnvironmentFGBFCS loadFGBFCSEnvironment()
        {
            FaultDetectionEnvironmentFGBFCS env = new
FaultDetectionEnvironmentFGBFCS();
            FGBWeb_General.General wsGeneral = new FGBWeb_General.General();
            wsGeneral.Url = ConfigurationManager.AppSettings.Get("FGB_WebServiceURL")
```

```
+ "general.asmx";
            FGBWeb_General.FaultDetectionEnvironment fdenv =
wsGeneral.FaultDetectionEnvironmentGet();
            foreach (FGBWeb_General.FaultDetectionEnvironmentAttribute attr in
fdenv.Attributes)
            {
                env.Attributes.Add(attr.Name, new FaultDetectionParam(attr.Value,
true));
            }
            return env;
        }
        /// <summary>
        /// Loads a FaultDetectionModuleTMCX module from a FGBWeb_TMC WebService
TMCTillInfoFaultDetection object.
        /// </summary>
        /// <param name="tmc">TMCTillInfoFaultDetection object representing the TMC to
create the Module object for</param>
        /// <returns>FaultDetectionModuleTMCX representing the
TMCTillInfoFaultDetection object given</returns>
        private static FaultDetectionModuleTMCX
loadTMCModuleFromTMCTillInfoFaultDetection(TMCTillInfoFaultDetection tmc)
        {
            FaultDetectionModuleTMCX moduleTMCX = new FaultDetectionModuleTMCX();
            moduleTMCX.Settings.Add("IsCashBackEnabled", new
FaultDetectionParam(tmc.IsCashBackEnabled.Equals("1").ToString(),
FaultDetectionParamType.BOOLEAN));
            moduleTMCX.Settings.Add("IsCashBackOnCashBackEnabled", new
FaultDetectionParam(tmc.IsCashBackOnCashBackEnabled.Equals("1").ToString(),
FaultDetectionParamType.BOOLEAN));
            moduleTMCX.Settings.Add("PosID", new FaultDetectionParam(tmc.PosID,
FaultDetectionParamType.STRING));
            moduleTMCX.Properties.Add("DB_IP", tmc.IP);
            moduleTMCX.Properties.Add("DB_MACAddress", tmc.MacAddress);
            moduleTMCX.Properties.Add("DB_LastOnline", tmc.LastOnline.ToString());
            return moduleTMCX;
        }
        /// <summary>
        /// Gets the TMC current state by using FGBWeb_TMCWeb web service.
        /// </summary>
        /// <param name="tillIP">the ipc of the TMC to get the state for</param>
        /// <returns>TMCResponse representing all the information that returned from
the TMC, including its current state</returns>
        private static FGBWeb_TMCWeb.TMCResponse TMCWSCallSimple(string tillIP)
        {
            FGBWeb_TMCWeb.TMCWeb wsTMCWeb = new FGBWeb_TMCWeb.TMCWeb();
            wsTMCWeb.Url = ConfigurationManager.AppSettings.Get("FGB_WebServiceURL") +
"tmcweb.asmx";
            try
            {
                return wsTMCWeb.GetStatusOfTMCByIP(tillIP);
            }
            catch
            {
                return null;
            }
        }
        /// <summary>
        /// Saves a recheck execution.
        /// </summary>
        /// <param name="tmc">the tmc the recheck was executed on</param>
        /// <param name="result">the results of the recheck</param>
        /// <param name="execution">the execution object from the saved execution on
```

```
the Fault Detection Knowledge-base</param>
        private static void saveTMCExecution(TMCTillInfoFaultDetection tmc,
FaultDetectionResult result, FaultDetectionExecution execution)
        {
            FGBWeb_TMC.TMC wsTMC = new TMC();
            wsTMC.Url = ConfigurationManager.AppSettings.Get("FGB_WebServiceURL") +
"tmc.asmx";
            wsTMC.TMCSaveFaultDetectionResult(tmc.MacAddress, (result.Executed ?
(result.Passed ? 0 : 1) : -1), result.Error + " [MANUAL]", (execution != null ?
execution.ID : -1));
        }
    }
}
```

3.4.2 TMCController.cs

An API Controller used to a list of all the TMCs or one specific TMC. Allows searching the list of TMCs for certain information (i.e. TMC versions, TMC IP ranges, etc.)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using System.Configuration;

namespace FGBFaultDetectionWebApplication.Controllers
{
    /// <summary>
    /// TMC Controller
    /// Used to a list of all the TMCs or one specific TMC.
    /// Allows searching the list of TMCs for certain information (i.e. TMC versions,
TMC IP ranges, etc.)
    /// </summary>
    public class TMCController : ApiController
    {
        // GET api/tmc
        /// <summary>
        /// Gets a list of all the TMCs from the FGBWeb_TMC Web Service.
        /// </summary>
        /// <returns>an array of TMCTillInfoFaultDetection object, each representing a
TMC in the current venue</returns>
        public FGBWeb_TMC.TMCTillInfoFaultDetection[] Get()
        {
            try
            {
                FGBWeb_TMC.TMC wsTMC = new FGBWeb_TMC.TMC();
                wsTMC.Url = ConfigurationManager.AppSettings.Get("FGB_WebServiceURL")
+ "tmc.asmx";
                return wsTMC.TMCGetFullDetailsList(null);
            }
            catch
            {
                return new FGBWeb_TMC.TMCTillInfoFaultDetection[0];
            }
        }

        // GET api/tmc/macaddress
        /// <summary>
        /// Gets a single TMCTillInfoFaultDetection object from the FGBWeb_TMC
WebService representing a specific TMC based on its mac address (identifier)
```

```
        /// </summary>
        /// <param name="identifier">the mac address of the TMC to get</param>
        /// <returns>TMCTillInfoFaultDetection representing the specific TMC (by its
mac address)</returns>
        public FGBWeb_TMC.TMCTillInfoFaultDetection Get(string identifier)
        {
            try
            {
                FGBWeb_TMC.TMC wsTMC = new FGBWeb_TMC.TMC();
                wsTMC.Url = ConfigurationManager.AppSettings.Get("FGB_WebServiceURL")
+ "tmc.asmx";
                return wsTMC.TMCGetByMacAddress(identifier);
            }
            catch
            {
                return new FGBWeb_TMC.TMCTillInfoFaultDetection();
            }
        }
        // GET api/tmc/?s=query
        /// <summary>
        /// Searchs all the TMCs in the current enviornment and returns a list of TMCs
which match search query (i.e. till name, pos id, mac address, ip addres, etc...)
        /// </summary>
        /// <param name="s">the search query to look for (i.e. can be TMC version
('v1.001.001'), a TMC IP range ('192.168.20.'), mac address, etc.)</param>
        /// <returns>a list of TMCs (represented by an array of
TMCTillInfoFaultDetection objects) of all the TMCs which match the search
query</returns>
        public FGBWeb_TMC.TMCTillInfoFaultDetection[] GetSearch([FromUri] string s)
        {
            try
            {
                FGBWeb_TMC.TMC wsTMC = new FGBWeb_TMC.TMC();
                wsTMC.Url = ConfigurationManager.AppSettings.Get("FGB_WebServiceURL")
+ "tmc.asmx";
                return wsTMC.TMCGetFullDetailsList(s);
            }
            catch
            {
                return new FGBWeb_TMC.TMCTillInfoFaultDetection[0];
            }
        }
    }
}
```

3.4.3 HomeController.cs

A Controller for the Index Page (i.e. the explanation facility web page). Returns a view based on "Views/Home/Index.cshtml" file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace FGBFaultDetectionWebApplication.Controllers
{
    /// <summary>
    /// Controller for the Index page (the Explanation Facility Page)
    ///
    /// Retruns a View based on Views/Home/Index.cshtml
```

```
    /// </summary>
    public class HomeController : Controller
    {

        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }

    }
}
```

Appendix C – Testing Documents

1. Testing Scenarios

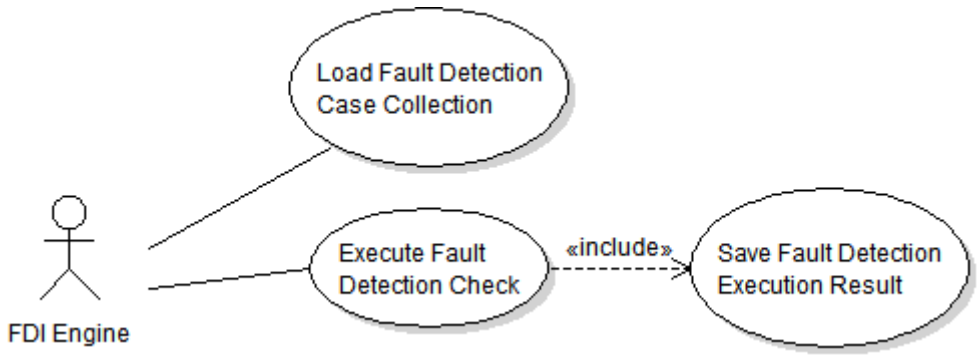
Testing scenarios were designed for each "sub-system" or actor in the solution. All scenarios start with the basic Use Cases, and expand from them to try and "break" the system.

All the scenarios below are included in their original Microsoft Excel format as part of Appendix D – Included Disc.

The following is the Base Use Cases and their derived scenario for each sub-system.

1.1 Fault Detection Engine

1.1.1 Base Use Cases



1.1.2 Testing Scenario

1	Setup / Basic			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
1.01	Open an Internet Browser and browse to the General.asmx file under the FGB_WebSrv you are working with	URL Found and Loaded		
1.02	Invoke the FaultDetectionEnvironmentGet call	Call executed and returned without error		
1.03	Launch the FaultDetectionTester app			
1.04	Set the "FGB Web Service URL" to the url of the			

	FGB_WebSrv you are working with (without the filename)			
1.05	Set the "Fault Detection DB connection String" to the correct Database			
1.06	Click the "Refresh List" button	the Available TMCs list gets populated.		
1.07	Click the "Detect Faults" button	The tester begins a fault detection test on the dummy TMCXStatus. The execution gets added to the History list The result pane gets populated with the results (Faults should be detected)		
1.08	Fill in the SMTP settings			
1.09	Using MS-SQL Server, connect to the Fault Detection database and change all notification teams email to yours.			
1.10	Check the "Send Notifications" checkbox and click "Detect Faults" again	The tester begins a fault detection test on the dummy TMCXStatus. The execution gets added to the History list The result pane gets populated with the results (Faults should be detected) Notifications were sent.		
1.11	Select a TMC you know is Online from the Available TMCs list	The TMC IP textbox gets populated with the selected TMC IP		

1.12	Check the "and Auto Detect" checkbox Click "Get Status"	The TMC Status string is loaded to the TMCXStatus Pane Fault Detection is executed on the selected TMC The Results pane gets populated with the results If faults were detected, notifications were sent		
1.13	Click "Detect All"	All the TMCs in the list get checked The results and status pane to not get updated If faults were detected, notifications were sent.		
1.14	Using an Internet Browser, browse to the tmc.asmx file under the FGB_WebSrv you are working with	URL Found and Loaded		
1.15	Invoke the TMCGetFullDetailsList (leave the search field empty)	A list of all the TMCs are loaded All TMCs XMLs have "Fault Detection" nodes Online TMCs have a FaultDetectionResult >= 0 When FaultDetectionResult >=0, FaultDetectionExecutionId has a positive value. FaultDetectionError and FaultDetectionLastUpdated fields have correct values.		

2	**Invalid URLs, DB, TMC Status JSON or SMTP Settings**			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
2.01	Close the FaultDetectionTester and Relaunch it			

2.02	Set the "FGB Web Service URL" to the url of the FGB_WebSrv you are working with (without the filename)			
2.03	Set the "Fault Detection DB connection String" to an ***incorrect*** Database			
2.04	Repeat Steps 1.06 to 1.15	No Fault Detections were executed. No Notifications were sent No Results were saved to the DB		
2.05	Set the "Fault Detection DB connection String" to a ***correct*** Database			
2.06	Set the "FGB Web Service URL" to an ***incorrect*** URL			
2.07	Repeat Steps 1.07 to 1.15	No Fault Detections were executed. No Notifications were sent No Results were saved to the DB		
2.08	Close the FaultDetectionTester and Relaunch it			
2.09	Set the "FGB Web Service URL" to the url of the FGB_WebSrv you are working with (without the filename)			
2.10	Set the "Fault Detection DB connection String" to the correct Database			
2.11	Edit the Raw TMCXStatus string, make sure its an invalid JSON			
2.12	Tick the "Send Notifications" checkbox and click "Detect Faults"	No Fault Detections were executed. No Notifications were sent No Results were saved to the DB		
2.13	Clear all the text in the Raw TMCXStatus string, make sure its an invalid JSON and click "Detect Faults" Again	No Fault Detections were executed. No Notifications were sent No Results were saved to the DB		
2.14	Repeat steps 2.08 to 2.10			

2.15	Set the SMTP Settings to invalid settings (i.e. incorrect username/password) and click "Detect Faults"	Fault detection executed, results panels were updated No notifications were sent		
2.16	Clear all the SMTP Settings fields, check "Send Notifications" checkbox and click "Detect Faults"	Fault detection executed, results panels were updated No notifications were sent		
2.17	Repeat steps 1.11 to 1.13	Fault detection executed, results panels were updated No notifications were sent		

3	**Invalid Rules**			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
3.01	Close the FaultDetectionTester App			
3.02	Using MS-SQL Server, connect to the FaulDetection Database			
3.03	Open the FaultDetection_Params table			
3.04	Edit the value of the "FaultDetection_ParamValue" column on one of the parameters with "FaultDetection_ParamType" of 20 (i.e. Module Attribute) to an invalid TMC Module Attribute			
3.05	Edit the value of the "FaultDetection_ParamValue" column on one of the parameters with "FaultDetection_ParamType" of 6 (i.e. Boolean) to "True1"			
3.06	Start the FaultDetectionTester app			
3.07	Set the "FGB Web Service URL" to the url of the FGB_WebSrv you are working with (without the filename)			
3.08	Set the "Fault Detection DB connection String" to the correct Database			

3.09	Click "Detect Faults" on the default TMC Status JSON	Fault detection executed, results panels were updated No notifications were sent In the results panel, the correct values were calculated for the parameters edited in steps 3.04 and 3.05. The Type 20 (Module Attribute) parameter was NULL The Type 6 (Boolean) was False.		
3.10	Repeat Steps 3.01 to 3.9 a few times - each time change different parameters in the Database.	Fault detection executed, results panels were updated No notifications were sent In the results panel, the correct values were calculated for all parameters. The Engine was able to deal with invalid values or types.		
3.11	Reset all Parameters in the Database to their original value			
3.12	Close the FaultDetectionTester App			

4	**Engine Simulation**			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
4.01	Start the FaultDetectionTester app			
4.02	Set the "FGB Web Service URL" to the url of the FGB_WebSrv you are working with (without the filename)			
4.03	Set the "Fault Detection DB connection String" to the correct Database			

4.04	Open "FaultFreeTMCX.json" file in Notepad, copy its content and paste into the tester in the Raw TMCXStatus textbox			
4.05	Click "Detect Faults"	Fault detection executed, results panels were updated No notifications were sent No Faults were detected		
4.06	Manually edit the JSON string in the Raw TMCXStatus text box, Change "Is Printer Set" value to "False"			
4.07	Click "Detect Faults"	Fault detection executed, results panels were updated No notifications were sent All the faults which are depended on the printer being set failed.		
4.08	Manually edit the JSON string in the Raw TMCXStatus text box, Reset "Is Printer Set" value to "True" Set "Is Connected" value to "False"	Fault detection executed, results panels were updated No notifications were sent All the faults which are depended on the printer being connected failed.		
4.09	Manually edit the JSON string in the Raw TMCXStatus text box, Reset "Is Connected" to True.			
4.10	Using MS-SQL connect to the Fault Detection Database, Open the FaultDetection_Cases and the FaultDetect_Tests tables			

4.11	For each one of the Tests in the DB: - change the corresponding Attribute in the Raw TMCXStatus text box of FaultDetectionTester App - Click "Detect Faults" - Reset the Attribute value to a "valid" one - Move to the next test. Note: you can look at FaultDetection_Tests_Rules to find out which attribute you need to change	Fault detection executed each time, results panels were updated No notifications were sent All the faults which are depended on the attribute that was changed failed.		
4.12	Open "FaultFreeTMCX.json" file in Notepad, copy its content and paste into the tester in the Raw TMCXStatus textbox			
4.13	Click "Detect Faults"	Fault detection executed, results panels were updated No notifications were sent No Faults were detected		

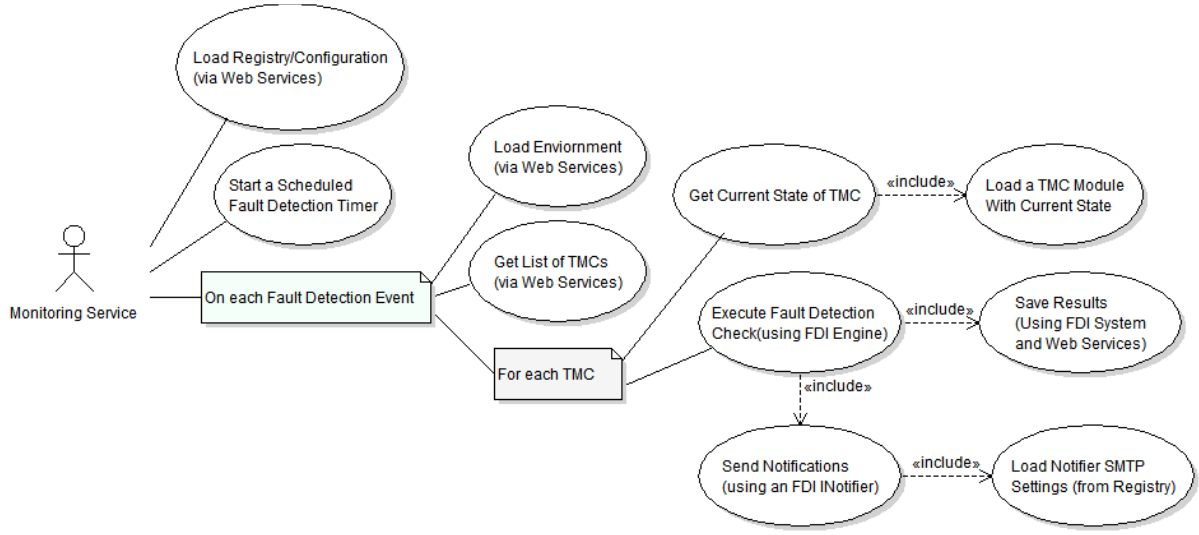
5	**Engine - Real TMC**			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
5.01	Start the FaultDetectionTester app			
5.02	Set the "FGB Web Service URL" to the url of the FGB_WebSrv you are working with (without the filename)			
5.03	Set the "Fault Detection DB connection String" to the correct Database			
5.04	Select the TMC you are working with from the Available TMCs List	The TMC IP textbox gets populated with the selected TMC IP		

5.05	Check the "and Auto Detect" checkbox Click "Get Status"	The TMC Status string is loaded to the TMCXStatus Pane Fault Detection is executed on the selected TMC The Results pane gets populated with the results No Faults were detected.		
5.06	Disconnect the Printer from the TMC, Wait 5-10seconds, Click "Get Status" on the FaultDetectionTester app	The TMC Status string is loaded to the TMCXStatus Pane Fault Detection is executed on the selected TMC The Results pane gets populated with the results All tests that are depended on the printer being connected failed.		
5.07	Reconnect the Printer to the TMC, Disconnect the RF Reader from the TMC, Wait 5-10seconds, click "Get Status" on the FaultDetectionTester app	The TMC Status string is loaded to the TMCXStatus Pane Fault Detection is executed on the selected TMC The Results pane gets populated with the results All tests that are depended on the reader being connected failed.		
5.08	Shutdown the DCS and/or SDS servers the TMC is connected to. Reconnect the Reader to the TMC Wait 5-10 seconds, click "Get Status" on the FaultDetectionTester App	The TMC Status string is loaded to the TMCXStatus Pane Fault Detection is executed on the selected TMC The Results pane gets populated with the results		

		All tests that are depended on the tic being online failed.		
5.09	While the DCS and/or SDS are closed, manually change the Key files on the TMC. Wait 5-10 seconds, click "Get Status" on the FaultDetectionTester App	The TMC Status string is loaded to the TMCXStatus Pane Fault Detection is executed on the selected TMC The Results pane gets populated with the results All tests that are depended on the tic being online and on the Encryption Keys failed.		
5.10	Restart the DCS and/or SDS server. Wait for the TMC to re-connect. Wait 5-10 seconds, click "Get Status" on the FaultDetectionTester App	The TMC Status string is loaded to the TMCXStatus Pane Fault Detection is executed on the selected TMC The Results pane gets populated with the results No Faults were detected - the TMC is online and the connection to the servers reset the keys to the correct ones.		

1.2 Fault Detection Monitoring Service

1.2.1 Base Use Cases



1.2.2 Testing Scenario

1	**Setup / Basic**			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
1.01	Open the services App.config file in notepad			
1.02	Set the FaultDetection_WebserviceURL key to the FGB_WebSrv you are working with.			
1.03	Set the xFGBFaultDetection connectionString to the Fault Detection Database you are working with.			
1.04	Using an Internet Browse, go to the General.asmx under the FGB_WebSrv you are working with	URL Found and Loaded		
1.05	Invoke the RegistryGet call with softwareId 148545 and true.	Call returned successfully.		
1.06	Using MS-SQL Server, connect to the FGBFCS you are working with and open the Registry Table, filter down by RG_SoftwareId = 148545	All settings found and have "valid" values. Make sure the FaultDetection_WebserviceURL and FaultDetection_FaultDetectionDBConnectionString		

		values match your configuration from steps 1.02 and 1.03		
1.07	Start the Service	Service Started, Event Log has logs indicating the service started and when the next time a Fault Detection will be executed. The FaultDetection_Log table in the Database is also updated.		
1.08	Stop the Service	Service Stopped, Event Log has logs indicating the service stopped . The FaultDetection_Log table in the Database is also updated.		
1.09	Using MS-SQL Server, connect to the FGBFCS you are working with and open the Registry Table, filter down by RG_SoftwareId = 148545 Change the value of FaultDetection_WebserviceURL to an invalid URL			
1.10	Start the Service	Service Starts and Stops immediately Event Log has logs indicating that the service failed to start due to invalid Web Service URL. The FaultDetection_Log table in the Database is also updated.		

1.11	Using MS-SQL Server, connect to the FGBFCS you are working with and open the Registry Table, filter down by RG_SoftwareId = 148545 Change the value of FaultDetection_WebserviceURL to an valid URL Change the value of FaultDetection_FaultDetectionDBConnectionString to and invalid Connection String.	Service Starts and Stops immediately Event Log has logs indicating that the service failed to start due to invalid Fault Detection DB Connection String. The FaultDetection_Log table in the Database is also updated.		
1.12	Using MS-SQL Server, connect to the FGBFCS you are working with and open the Registry Table, filter down by RG_SoftwareId = 148545 Change the value of FaultDetection_FaultDetectionDBConnectionString to and valid Connection String. Change the value of FaultDetection_IntervalType to an invalid value (anything other then 1, 2 or 3). Change the value of FaultDetection_IntervalValue to an invalid value (-1)	Service Starts and Stops immediately Event Log has logs indicating that the service failed to start due to invalid Interval Type or Interval Value. The FaultDetection_Log table in the Database is also updated.		
1.13	Using MS-SQL Server, connect to the FGBFCS you are working with and open the Registry Table, filter down by RG_SoftwareId = 148545 Change the value of FaultDetection_IntervalType to an valid value (1, 2 or 3). Change the value of FaultDetection_IntervalValue to an invalid value (-1)	Service Starts and Stops immediately Event Log has logs indicating that the service failed to start due to invalid Interval Type or Interval Value. The FaultDetection_Log table in the Database is also updated.		

1.14	Using MS-SQL Server, connect to the FGBFCS you are working with and open the Registry Table, filter down by RG_SoftwareId = 148545 Change the value of FaultDetection_IntervalValue to an valid value (60)	Service Started, Event Log has logs indicating the service started and when the next time a Fault Detection will be executed. The FaultDetection_Log table in the Database is also updated.		
1.15	Stop the Service	Service Stopped, Event Log has logs indicating the service stopped . The FaultDetection_Log table in the Database is also updated.		

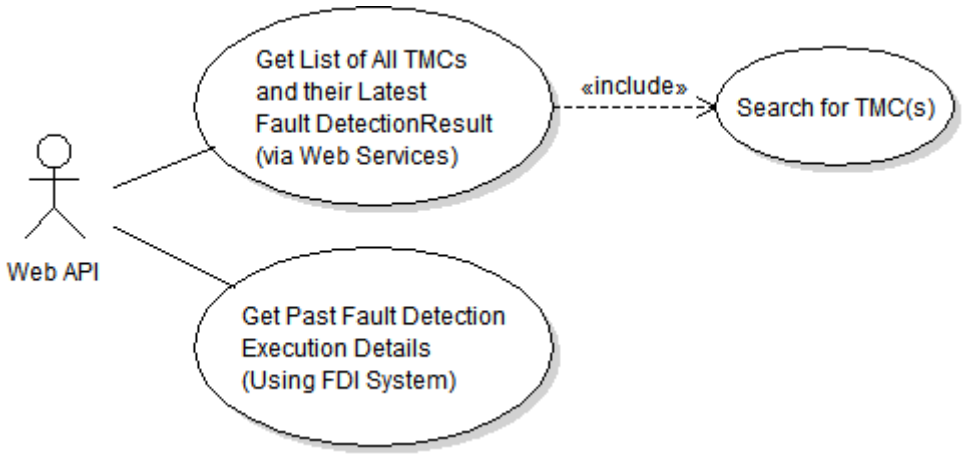
2	**Connection Errors**			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
2.01	Using MS-SQL Server, connect to the FGBFCS you are working with and open the Registry Table, filter down by RG_SoftwareId = 148545 Change the value of FaultDetection_IntervalType to 3 (minute) Change the value of FaultDetection_IntervalValue to 1	Service Started, Event Log has logs indicating the service started and when the next time a Fault Detection will be executed. The FaultDetection_Log table in the Database is also updated.		
2.02	Go to IIS Manager and Stop the FGB_WebSrv application you are working with Wait 1 minute for the Interval to expire and a Fault Detection Execution to be triggered	Fault Detection Skipped. Event Log and FaultDetection_Log table are up-to-date Service did not crash and is still working		

2.03	Go to IIS Manager and Start the FGB_WebSrv application you are working with Wait 1 minute for the Interval to expire and a Fault Detection Execution to be triggered	Fault Detection Executed. Results are saved and if needed, notifications were sent.		
2.04	Disconnect the TMC your working with from the Network. Wait 1 minute for the interval to expire and a Fault Detection Execution to be triggered	Fault Detection Executed. Results are saved and if needed, notifications were sent. The Fault Detection was skipped on TMC you disconnected, and its shown in the FaultDetection_Log and in the TMC_FaultDetection_LastResult tables.		
2.05	Reconnect the TMC your working with from the Network. Wait 1 minute for the interval to expire and a Fault Detection Execution to be triggered	Fault Detection Executed. Results are saved and if needed, notifications were sent. The Fault Detection was executed on TMC you disconnected, and its shown in the FaultDetection_Log and in the TMC_FaultDetection_LastResult tables.		
2.06	Disconnect the Printer from the TMC your working with. Wait 1 minute for the interval to expire and a Fault Detection Execution to be triggered	Fault Detection Executed. Results are saved and if needed, notifications were sent. The Fault Detection was executed and faults were detected on TMC you disconnected the printer from. It is shown in the FaultDetection_Log and in the		

		TMC_FaultDetection_LastResult tables.		
2.07	Reconnect the Printer from the TMC your working with. Wait 1 minute for the interval to expire and a Fault Detection Execution to be triggered	Fault Detection Executed. Results are saved and if needed, notifications were sent. The Fault Detection was executed and not faults were found for your TMC. Its shown in the FaultDetection_Log and in the TMC_FaultDetection_LastResult tables.		

1.3 Fault Detection Web API

1.3.1 Base Use Cases



1.3.2 Testing Scenario

1	**Setup / Basic**			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
1.01	Open the WebAPI Web.config file in notepad			
1.02	Set the FGB_WebserviceURL key			

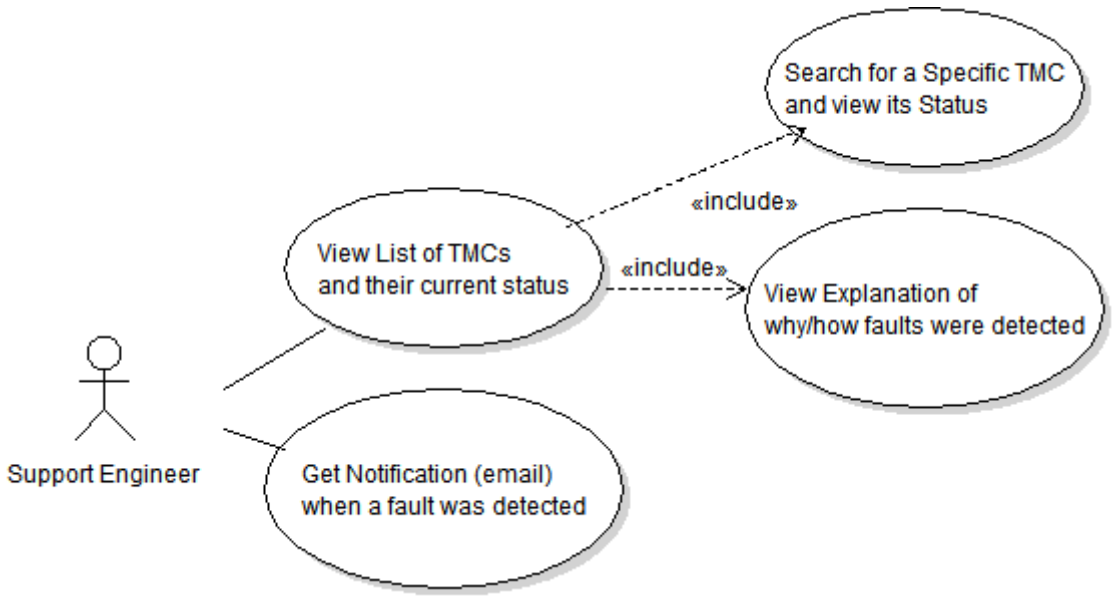
	to the FGB_WebSrv you are working with.			
1.03	Set the xFGBFaultDetection connectionString to the Fault Detection Database you are working with.			
1.04	Using an Internet Browse, go to /api/tmc/ under the Web API you are working with	A JSON string representing a list of all TMCs is returned Each TMC has its latest Fault Detection Status attached to it.		
1.05	Using an Internet Browse, go to /api/tmc/?s=<IP OF YOUR TMC>	A JSON string representing a list of all TMCs is returned It only has 1 TMC (matching the IP entered) and its latest Fault Detection Status attached to it.		
1.06	Using an Internet Browse, go to /api/tmc/?s=12344324	A JSON string representing a list of all TMCs is returned It is am empty array as no TMCs were found		
1.07	Using an Internet Browse, go to /api/tmc/?s=<IP OF YOUR TMC> When the JSON string returns, copy the value of the "FaultDetectionExecutionId"			
1.08	Using an Internet Browse, go to /api/faultdetection/	A JSON string representing an error is returned.		
1.09	Using an Internet Browse, go to /api/faultdetection/<The Execution Id from Step 1.07>	A JSON string representing the Fault Detection Execution and all the tests that were done is returned.		

1.10	Using an Internet Browse, go to /api/faultdetection/<The Execution Id from Step 1.07>/1	A JSON string representing the Fault Detection Execution and all the tests that were done is returned. This time, only cases/tests which failed are returned		
1.11	Using an Internet Browse, go to /api/faultdetection/1234567/	A JSON string representing an empty Fault Detection Execution is returned. FaultCollection is null.		
1.12	Using an Internet Browse, go to /api/faultdetection/1234567/1	A JSON string representing an empty Fault Detection Execution is returned. FaultCollection is null.		
1.13	Open the WebAPI Web.config file in notepad			
1.14	Set the FGB_WebserviceURL key to the an invalid URL.			
1.15	Repeat steps 1.04 to 1.6	A JSON string representing a list of all TMCs is returned An empty array as no TMCs were found was always returned		
1.16	Open the WebAPI Web.config file in notepad			
1.17	Set the FGB_WebserviceURL key to the FGB_WebSrv you are working with.			
1.18	Set the xFGBFaultDetection connectionString to an invalid Database			

1.19	Repeat steps 1.08 to 1.12	Every time a JSON string representing an empty Fault Detection Execution is returned. FaultCollection is null. Calls take slightly longer because the connect to the DB fails.		
1.20	Open the WebAPI Web.config file in notepad			
1.21	Set the FGB_WebserviceURL key to the FGB_WebSrv you are working with.			
1.22	Set the xFGBFaultDetection connectionString to the Fault Detection Database you are working with.			

1.4 Fault Detection Explanation Facility

1.4.1 Base Use Cases



1.4.2 Testing Scenario

1	**Setup / Basic**			
Step	Action	Expected Result	Pass/Fail	Defect/Actual Result
1.01	Open the WebAPI Web.config file in notepad			

1.02	Set the FGB_WebserviceURL key to the FGB_WebSrv you are working with.			
1.03	Set the xFGBFaultDetection connectionString to the Fault Detection Database you are working with.			
1.04	Using an Internet Browse, go to / (root) of the Web API you are working with	A Web Page is loaded It has a Search box, a button one the right side "Displaying Faulty Only" and a list of Faulty TMCs (if found)		
1.05	Click the "Displaying Faulty Only" button	The button changes colours to grey, the checkbox becomes clear. A complete list of TMCs is displayed: TMCs with faults are coloured Red TMCs that were not checked are coloured Orange TMCs that did not have any faults are coloured White		
1.06	Using the Search box, enter the IP of the TMC your working with, click the "Refresh" button	A loading Panel comes up After a couple of seconds the list is refreshed and only 1 TMC is displayed, matching your search.		
1.07	Using the Search box, enter the 12343543, click the "Refresh" button	A loading Panel comes up After a couple of seconds a message saying "No TMCs found to display" is shown.		

1.08	Clear the text in the search box and click "Refresh"	A loading Panel comes up After a couple of seconds the original list from step 1.05 is shown.		
1.09	Click the "Displaying Faulty Only" button	The button changes colours to green the checkbox becomes checked A list of only faulty TMCs is displayed		
1.10	Click on one of the "Exclamation Mark" buttons of one of the Faulty TMCs	A loading panel comes up. After a few seconds the loading panel disappears and a JQuery Dialog comes up with the results of the Fault Detection Execution. Only Failed Checks are displayed and they are marked in Red. The execution ID and the time it was done is clearly displayed.		
1.11	Open the WebAPI Web.config file in notepad			
1.12	Set the FGB_WebserviceURL key to the an invalid URL.			
1.13	Repeat steps 1.04 to 1.09	Each time a message saying "No TMCs Found to Display" is shown.		
1.14	Open the WebAPI Web.config file in notepad			
1.15	Set the FGB_WebserviceURL key to back to the URL of the FGB_WebSrv you are working with. Set the xFGBFaultDetection			

	connectionString to the an invalid URL			
1.16	Repeat steps 1.04 to 1.09	Expected Results are the same, a list of TMCs is displayed according to the step.		
1.17	Repeat step 1.10	A loading panel comes up. The loading panel stays up for a little longer, as the connection to the DB is failed. After it is gone, the same JQuery Dialog comes up, however, this time an error message saying "Failed getting Fault Detection Execution (Id: ###)" is shown.		
1.18	Open the WebAPI Web.config file in notepad			
1.19	Set the xFGBFaultDetection connectionString to the Fault Detection DB your working with.			
1.20	Using MS-SQL connect to the FGBFCS your working with, Open the TMC_FaultDetection_LastResult table, and find the TMC your working with Note: you can find the MAC address for your TMC in the TMC_TillInfo table	The TMC is found in the Table		
1.21	Change the FaultDetection_Result to 1 and FaultDetection_ExecutionID to 1234567			
1.22	Reload the Web Explanation Facility page, using the search box search for your TMC by IP	The list gets filtered down, the TMC is shown to have a Fault		

1.23	Click on one of the "Exclamation Mark" buttons of your TMC	A loading panel comes up. After a couple of seconds its gone, the same JQuery Dialog comes up, however, this time an error message saying "Failed getting Fault Detection Execution (Id: 1234567)" is shown.		
1.24	Clear the search box, find another faulty TMC and click on its "Exclamation Mark" button	A loading panel comes up. After a few seconds the loading panel disappears and a JQuery Dialog comes up with the results of the Fault Detection Execution. Only Failed Checks are displayed and they are marked in Red. The execution ID and the time it was done is clearly displayed.		
1.25	Using MS-SQL connect to the FGBFCS your working with, Open the TMC_FaultDetection_LastResult table, and find the TMC your working with and change the values back to their original value.			

1.26	Reload the Web Explanation Facility page, find a faulty TMC and click on its "Exclamation Mark" button	A loading panel comes up. After a few seconds the loading panel disappears and a JQuery Dialog comes up with the results of the Fault Detection Execution. Only Failed Checks are displayed and they are marked in Red. The execution ID and the time it was done is clearly displayed.		
1.27	On the Fault Detection Result dialog, click on the Recheck button (If nothing changed in the TMC, fault should be detected. If no fault is detected, use a different TMC)	A loading panel comes up and stays for a few seconds. When it disappears, the dialog is refreshed with the new results, this time all checks are listed, not only ones that failed. Remember the new fault detection execution ID		
1.28	Close the fault detection dialog			
1.29	If fault has been detected, click on the TMC Exclamation Mark again to re-open the Fault Detection Dialog.	The dialog re-opens. The fault detection execution ID is the same as what returned in step 1.27, only failed checks are displayed.		

1.30	Refresh the Explanation Facility web page, find the same TMC again and click the "Exclamation Mark" again.	A loading panel comes up. After a few seconds the loading panel disappears and a JQuery Dialog comes up with the results of the Fault Detection Execution. Only Failed Checks are displayed and they are marked in Red. The execution ID matches the execution ID from step 1.27		
1.31	Physically disconnect the TMC your working with from the network. Click Recheck button while the TMC is disconnected	A loading panel comes up. After a couple of seconds its gone, the same JQuery Dialog comes up, however, this time an error message saying "Failed getting recheck result…" asking the user to verify that the TMC is connect to the network.		
1.32	Reconnect the TMC your working with from the network. Close the dialog, and click the "exclamation mark" button again - when the dialog re-appears, Click Recheck button while the TMC is disconnected	A loading panel comes up and stays for a few seconds. When it disappears, the dialog is refreshed with the new results, this time all checks are listed, not only ones that failed.		

2. EngineTester Module Fault Cases and Tests

The EngineTester module was used to check all possible parameter types and the operation types they are valid for.

Fault Case	Tests	Dummy JSON and KB Parameter Values
Validate Boolean Parameters/Operations	1. Boolean True is True 2. Boolean False is False 3. Boolean True is Not False 4. Boolean False is Not True	"BTrue" : "True", "BFalse" : "False"
Validate String Parameters/Operations	1. String1 Equals String1 2. String12 Contains String1 3. String12 Longer Than String1 4. String1 Shorter Than String12 5. String1 Longer Equals String1 6. String1 Shorter Equals String1 7. String1 Not Contains String12 8. String12 Equals Regex.oneone 9. Regex.oneone Equals String12	"String1" : "one", "String12" : "oneone" Regex.onone: “(?:one){2}”
Validate Integer Parameters/Operations	1. Int1 Equals Int1 2. Int1 Less Than Int2 3. Int2 Greater Than Int1 4. Int1 Not Equals Int2 5. Int1 Not Equals Decimal101 6. Int1 Less Than Decimal101 7. Int2 Larger Than Decimal101 8. Int1 Equals Decimal100	"Int1": "1", "Int2": "2", "Decimal101": "1.01", "Decimal100": "1.00"
Validate Decimal Parameters/Operations	1. Decimal101 Equals Decimal101 2. Decimal101 Smaller Than Decimal102 3. Decimal102 Greater Than Decimal101 4. Decimal100 Equals Int1	"Decimal101": "1.01", "Decimal102": "1.02", "Int1": "1"
Validate StringArrays Parameters/Operations	1. SA123 Equals SA123 2. SA123 Shorter Than SA1234 3. SA1234 Longer Than SA123 4. SA123 Contains String1 5. SA123 Not Contains String12 6. SA1234 Contains Regex.ThreeOrFour	"SA123": "one,two,three", "SA1234": "one,two,three,four" Regex.ThreeOrFour: “three|four”
Validate IntArrays Parameters/Operations	1. IA123 Equals IA123 2. IA123 Not Equals IA2345 3. IA123 Shorter Than IA2345 4. IA2345 Longer Than IA123 5. IA123 Contains Int1 6. IA2345 Not Contains Int1	"IA123": "1,2,3", "IA2345": "2,3,4,5"
Validate DecimalArrays Parameters/Operations	1. DA10123 Equals DA10123 2. DA10123 Not Equals DA102345 3. DA10123 Shorter Than	"DA10123": "1.01,1.02,1.03", "DA102345": "1.02,1.03,1.04,1.05"

	DA102345 4. DA102345 Longer ThanDA10123 5. DA10123 Contains Decimal101 6. DA102345 Not Contains Decimal101	
Validate Dates Parameters/Operations	1. Date2014-04-28 Equals Date2014-04-28 2. Date2014-04-28 Less Than Date2014-05-01 3. Date2014-05-01 Greater Than Date2014-04-28	"Date2014-04-28": "28/04/2014 00:00:00", "Date2014-05-01": "01/05/2014 00:00:00"
Validate TimeSpans Parameters/Operations	1. TS24HR Equals TS1DAY 2. TS50MIN Shorter Than TS1HR 3. TS1DAY Longer Than TS1HR	"TS50MIN": "50 minutes", "TS1HR": "1 hour", “TS24HR”: “24 hours”, "TS1DAY": "1 day"

Appendix D – Included Disc

A CD has been attached to the report. It contains:

- The Source Code of the Fault Detection Engine (FDI System), the Monitoring Service (and its Kernel), the Web API and Web Explanation Facility.
- Compiled versions of the Fault Detection Engine (FDI) System DLL, the Monitoring Service (and its Kernel), the Web API and Web Explanation Facility.
- Copies of the Testing Scenarios from Appendix C in their original Microsoft Excel format.
- The Class Diagram from Appendix B1.1 in its original Microsoft Visio format (VSD).