



PRODUCTIVITY
ENHANCEMENT
SYSTEMS

Embedded Diagnostics with TestBase

White Paper

November 2002

Introduction

TYX TestBase [1] is a software product that enables the visual development and run-time execution of diagnostic strategies utilizing large bodies of test procedures. Its flexibility allows modification of parametric data without having to resort to modification of source code. Currently, it supports test and diagnostics in production (factory) and maintenance environments.

This paper describes the expansion of TestBase into a third environment - the *run-time execution of test and diagnostics on embedded systems*. With this new functionality, systems and system components can be constantly monitored and diagnosed in real-time, allowing mission commanders and operators an on-going view of the health of their systems.

Used together, the existing TestBase and the newer Embedded TestBase provide *integrated test and diagnostics capabilities* for all three environments.

Embedded TestBase is provided as a standalone product and is currently available as a prototype.

Functionality and Architecture

Figure 1 shows the architecture of an *integrated test and diagnostics solution* supported by TestBase, encompassing the three operational environments:

- The **Diagnostic Development Environment** contains the software applications that enable the development of diagnostic strategies to be executed in the Maintenance and Embedded Environments.
- The **Maintenance Environment** contains test and diagnostics software, test instrumentation and data storage components that support the testing, diagnosis and maintenance of the vehicle off-line between missions.
- The **Embedded Environment** contains the vehicle hardware and software, including the software components that implement test and diagnostics on-line and in real-time during a mission.

Diagnostic Development Environment

The development of diagnostic strategies is performed using the **TestBase Integrated Development Environment (IDE)** [1], which enables the visual design of “fault tree” strategies. The diagnostic strategy information, including sequence information and parametric data, is stored in a **TestBase Database**.

The TestBase IDE is capable of importing diagnostic strategy information from **Third-Party Diagnostic Development Software**, such as DSI eXpress [2] and Intusoft Test Designer [3]. These applications generate diagnostic strategies using vehicle design data in the form of

CAD/CAEE files, simulation models and human expertise. The transfer of diagnostic strategy data is performed via an XML-based format called **DiagML**, which was jointly developed by TYX and DSI International and is offered as an open industry specification.

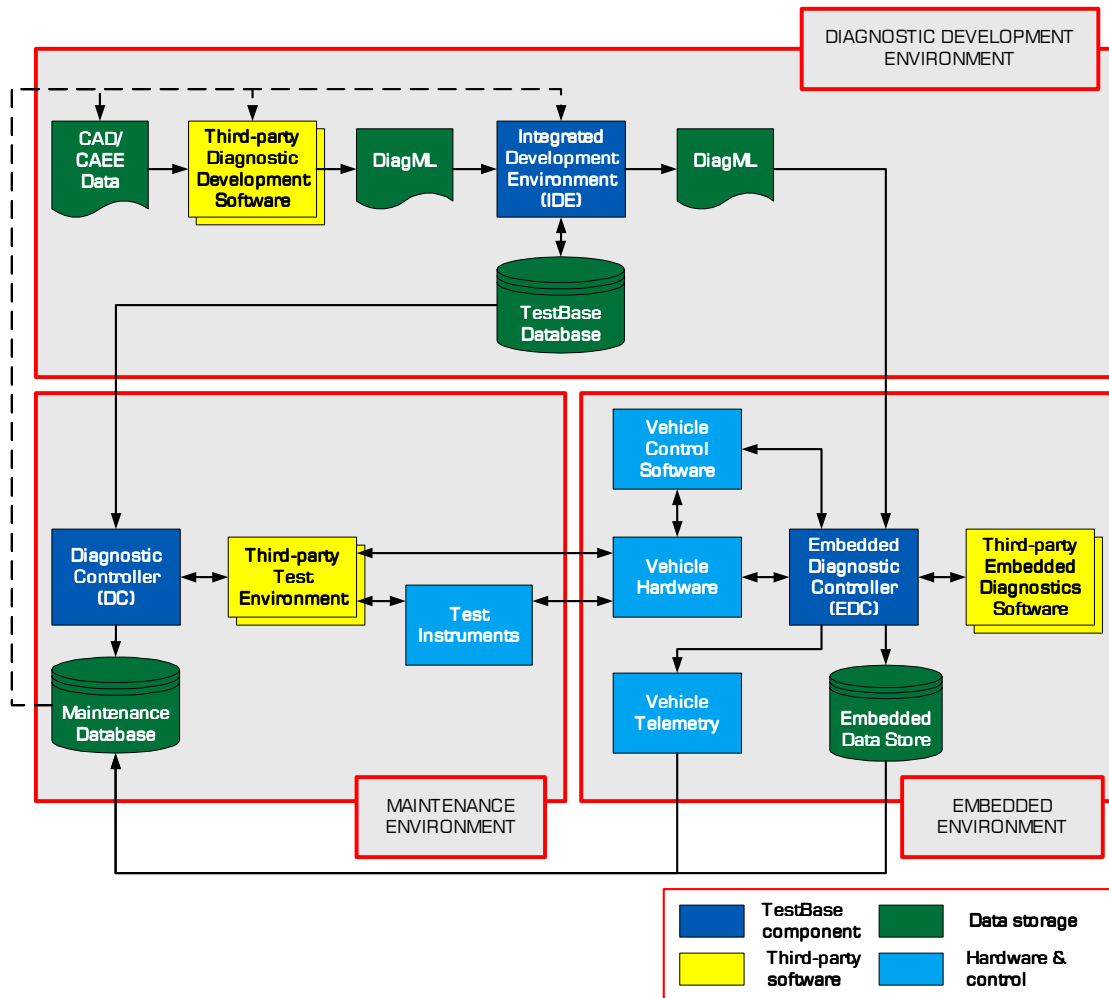


Figure 1 - TestBase Integrated Diagnostics Architecture

Maintenance Environment

The execution of diagnostic strategies in the Maintenance Environment is performed by the **TestBase Diagnostic Controller (DC)**, which retrieves sequence information and parametric data from the TestBase Database.

The DC triggers the execution of external software modules, called “test procedures”, according to the sequence information from the TestBase Database. Test procedures receive parametric data from the DC which has, in turn, retrieved them from the TestBase Database. Test procedures return an outcome to the DC indicating the result of the test, and possibly other test results such as measurement values.

The test procedures are developed and executed with various **Third-party Test Environments**. TestBase supports specialized test applications such as VEE, LabVIEW, LabWindows/CVI and PAWS ATLAS, as well as general-purpose programming languages including C, C++ and Visual

Basic. The test procedures may use **Test Instruments**, as well as a direct communication channel to the **Vehicle Hardware**, used to transmit commands and receive sensor and status data.

At the end of a diagnostic strategy execution, the DC determines a diagnostic conclusion, representing a fault or a fault group.

The DC stores test and diagnostics results (parametric data passed to test procedures, outcomes and other results returned by test procedures, diagnostic conclusions, etc.) in a **Maintenance Database**. TestBase currently supports relational databases and an XML-based file format. The Maintenance Database may store additional maintenance-related information input by maintenance personnel, such as the faults identified during repair, the duration and cost of repair operations, etc.

Historical data from the Maintenance Database may be used to improve the vehicle design, by redesigning the components that cause the most frequent failures or by adding redundancy. In addition, Maintenance Database data may be used to improve the diagnostics design, in order to maximize fault detection and isolation capabilities. Lastly, the statistical analysis of Maintenance Database information may support the optimization of maintenance operations.

Embedded Environment

The execution of test and diagnostics operations in the Embedded Environment is coordinated by the **TestBase Embedded Diagnostic Controller (EDC)**. While this component provides the same basic functionality as the DC used in the Maintenance Environment, its implementation is optimized for embedded execution. This optimization meets the following objectives:

- portability of implementation to various embedded operating systems (OSs)
- reduced resource requirements in terms of memory utilization and computational load
- capability to interact with the vehicle control software
- capability to interact with third-party embedded diagnostics software

To avoid the high resource consumption of a database engine, diagnostic strategy data is uploaded from the TestBase IDE to the Embedded Environment using **DiagML** or an application-specific data format generated from DiagML.

The EDC operates under the supervision of the **Vehicle Control Software**, which requests the execution of diagnostic strategies corresponding to various control modes. The EDC receives sensor data and state information and returns diagnostic conclusions. In addition, the EDC may send command requests, instructing the Vehicle Control Software to stimulate the Vehicle Hardware in order to facilitate diagnosis. The EDC is also capable of interacting directly with the **Vehicle Hardware**, via application-specific software routines (the embedded equivalent of the test procedures used in the Maintenance Environment).

The sensor data, test results and diagnostic conclusions are stored by the EDC in an **Embedded Data Store** (database, file or memory), for download at the end of the mission. Alternatively, this information may be transmitted during the mission, via the **Vehicle Telemetry** subsystem. In both cases, the data is ultimately stored in the **Maintenance Database**. Diagnostics information received from the Embedded Environment via telemetry may be used for advance preparation of maintenance operations to be performed at the end of the mission. In addition, it may be used by the TestBase DC as a starting point for maintenance diagnosis. Besides a reduction of diagnostics time, this approach has the potential of providing better diagnostic performance by using sensor data captured during the actual occurrence of faults.

In addition, the EDC is capable of interacting with **Third-Party Embedded Diagnostics Software** that implements dynamic fault detection and isolation, such as model-based reasoning tools, rule-based expert systems, neural networks, signal analysis tools, etc.

Embedded Diagnostic Controller

Architecture

Embedded diagnostic applications have vastly different requirements in terms of functionality, hardware resources (memory, computing power, hardware interfaces), software environments (operating system, programming languages, software interfacing technologies) and quality management procedures. Consequently, the EDC was designed to *enable multiple implementations, while preserving compatibility with the TestBase product.*

The internal architecture of the Embedded DC, shown in Figure 2, includes:

- *Common modules*, portable across different Operating Systems. These modules are developed and distributed by TYX as part of the Embedded TestBase product.
- *Custom modules*, which may be OS-specific and are typically developed for each particular application. These modules tailor the operation of the EDC for the application, including support for a specific operating system, data formats and executable code formats.

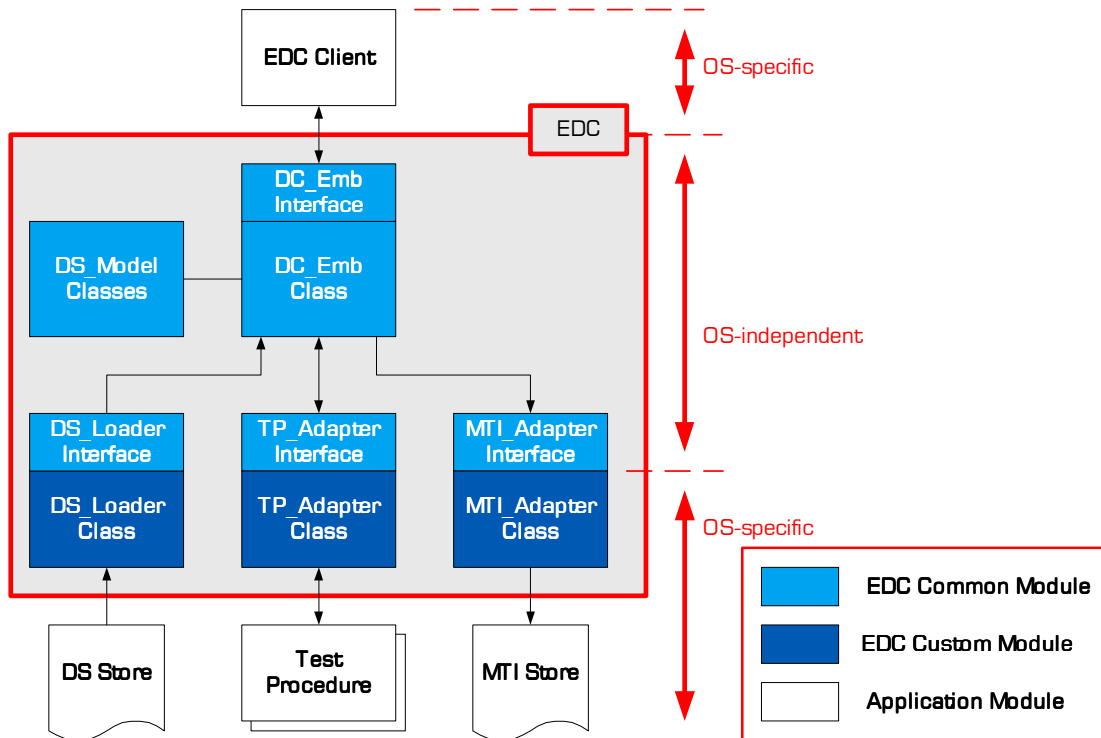


Figure 2 – Internal EDC Architecture

The **EDC Client** may be any application module that requires the diagnostic services of the EDC. Typically, this would be the vehicle control software. The EDC Client requests the EDC to execute a Diagnostic Strategy and in response receives diagnostic conclusions.

The **DC_Emb Class** is the core of the EDC, implementing test sequencing functionality and handling the transfer of parametric data. The **DC_Emb Interface**, specified by the Embedded TestBase documentation, defines the functionality available to **EDC Clients**. The interface provides methods that load and unload a Diagnostic Strategy, execute the currently loaded Diagnostic Strategy and abort the execution. It also supports callbacks that enable Test Procedures to send messages to the EDC Client.

The **DS_Model Classes** are used by the other classes to build and access an internal model of the loaded Diagnostic Strategy.

The interfaces **DS_Loader Interface**, **TP_Adapter Interface** and **MTI_Adapter Interface** specified by the Embedded TestBase documentation define the functionality that must be implemented by the custom modules, in order to interoperate with the common modules.

The **DS_Loader Class** reads Diagnostic Strategy data from an application-specific **DS Store** (such as an XML file, a binary file, a database or an upload communication channel) and builds an internal Diagnostic Strategy model. The **DS_Loader Interface** contains a method that loads a Diagnostic Strategy.

The **TP_Adapter Class** performs the execution of external Test Procedures that have an application-specific format, such as: function in a binary library or component, scripting language, intermediate language executed by a run-time engine, etc. The **TP_Adapter Interface** contains a method that executes one Test Procedure. The interface also supports callbacks that enable the Test Procedures to send messages to the EDC Client.

The **MTI_Adapter Class** writes test and diagnostic results to an application-specific **MTI Store** (such as an XML file, a binary file, a database or a download communication channel). The **MTI_Adapter Interface** contains methods that store diagnostic conclusions, test outcomes and test parametric data.

Third-party Embedded Diagnostic Software may be integrated with the EDC using several architectural solutions. For example, such software may operate in a master role as an EDC Client, in a slave role as a special Test Procedure, or in parallel, in which case its interaction with the EDC is managed by the Vehicle Control Software.

Customization and Compatibility

The customization of Embedded TestBase for a particular application typically includes the following tasks:

- Develop the EDC custom modules DC_Loader Class, TP_Adapter Class and MTI_Adapter Class, implementing the interfaces DC_Loader Interface, TP_Adapter Interface and MTI_Adapter Interface defined by the Embedded TestBase documentation.
- Develop the EDC Client code that interacts with the EDC via the DC_Emb Interface.
- Compile the EDC common components for the target OS.
- Link the EDC common and custom components in the embedded software code.

The *common modules* are maintained by TYX and are guaranteed to be compatible with newer versions of the TestBase IDE.

The *custom modules* may be developed by TYX or by a third party. As long as these modules implement the interfaces and the functionality originally specified by the Embedded TestBase documentation, they are guaranteed to be compatible with all future versions of the common modules.

Embedded TestBase Prototype

Embedded TestBase is available as a prototype that includes common components written in ANSII C++ and custom components targeting Microsoft Embedded Windows XP. The prototype may be evaluated using Embedded Windows XP, or a desktop version of Windows such as Windows NT 4.0, Windows 2000 or Windows XP.

Architecture

The architecture of the prototype is shown in Figure 3. Implementation details for its modules are provided in the next section.

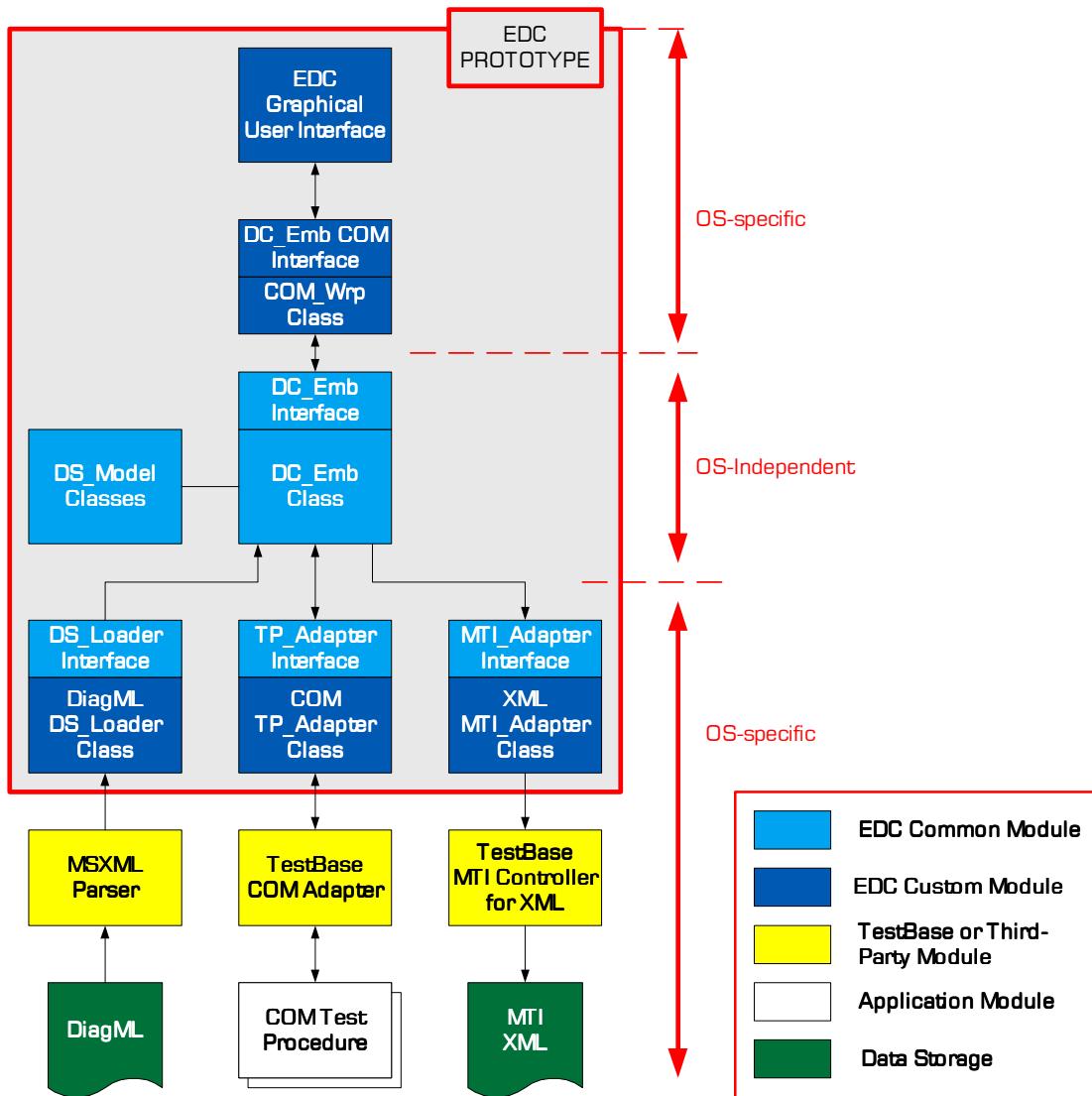


Figure 3 - Architecture of the Embedded TestBase Prototype

Implementation

The *common modules* of the EDC prototype are written in ANSI C++. TYX plans to develop and maintain several versions of the common modules written in different languages, including ANSI C.

The *custom modules* of the prototype target Embedded Windows XP. This choice enables the use of existing TestBase and third-party components that operate on the Windows platform, as well as Windows-specific Rapid Application Development technologies. As indicated before, this does not preclude alternative implementations that target other operating systems and use software technologies with lower resource requirements.

The **EDC Graphical User Interface**, used for demonstration purposes, controls the loading and execution of diagnostic strategies and displays diagnostic results. This component is developed with Visual Basic and is available in source format.

The **COM_Wrapper Class** is a wrapper that exposes a COM version of the **DC_Emb Interface**, facilitating usage from clients developed with several programming languages, including Visual Basic.

The **DiagML_DS_Loader Class** supports the loading of Diagnostic Strategy data from the DiagML format. Its implementation uses Microsoft's MSXML Parser.

The **COM_TP_Adapter Class** performs the execution of Test Procedures implemented as COM components. The implementation of the class uses the "TestBase COM Adapter", which is also part of the regular TestBase distribution.

The **XML_MTI_Adapter Class** supports the storage of results in an XML-based format. Its implementation uses the "TestBase MTI Controller for XML", which is also part of the regular TestBase distribution.

The Embedded TestBase distribution also contains a set of ready-to-run *samples*, including TestBase Databases, DiagML files and COM Test Procedures (also available in source format).

Conclusions

Embedded TestBase is a version of the TestBase run-time environment specifically designed for operation on embedded systems. The modular architecture of Embedded TestBase isolates application-specific functionality in separate modules, *facilitating re-hosting and adaptation to the particularities of each embedded application*.

The development environment of TestBase supports the *integrated development of diagnostics for embedded, production and maintenance environments*. In this approach, different diagnostic strategies, supporting the different functional requirements of these environments, may be derived from a unique set of design and diagnostic models.

TestBase and Embedded TestBase enable the *integrated storage of test and diagnostics results generated in the embedded and maintenance environments*. This solution increases the efficiency of maintenance diagnostics and supports the optimization of maintenance operations. Stored data may be used for improvements in vehicle design and manufacturing, as well as the optimization of diagnostics design.

The open architecture of TestBase and Embedded TestBase enables their *integration with third-party diagnostics software*, in both design and run-time environments. The modularity of Embedded TestBase enables it to operate as the *central point of integration*, between multiple third-party diagnostic applications.

Glossary of Terms and Acronyms

CAD: Computer-Aided Design

CAEE: Computer-Aided Electrical Engineering

DC: Diagnostic Controller

diagnostic strategy: software representation of a method, algorithm, etc., that performs a diagnostics operation

diagnostics: operation that detects and isolates faults in a system

EDC: Embedded Diagnostic Controller

IDE: Integrated Development Environment

MTI: Maintenance Test Information

OS: Operating System

test: operation that verifies that the characteristics of a system are within the limits characterizing its nominal operation

test procedure: software implementation of a method, algorithm, etc., that performs a **test** operation

XML: eXtensible Markup Language

References

- [1] TYX TestBase, TYX Web site, <http://www.tyx.com/testbase.html>
- [2] DSI eXpress, DSI International Web site, <http://www.dsiintl.com/Products/index.asp>
- [3] Intusoft Test Designer, DSI International Web site, <http://www.intusoft.com/products/testdesigner.htm>