Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

STDF2EXF Converter Development at Critical Software, SA

MIEIC Project Report 2007/2008

Daniel José Santos da Silva

Supervisor from FEUP: Prof.ª Ana Paula Rocha

Supervisor from Institution: Hugo Casimiro

March 2008

Summary

This document reports the work made by Daniel José Santos da Silva in the company Critical Software, SA, within the scope of the Project from the Integrated Master degree in Informatics and Computing Engineering (*Mestrado Integrado em Engenharia Informática e Computação* – MIEIC) at the University of Porto, Faculty of Engineering (*Faculdade de Engenharia da Universidade do Porto* – FEUP), which took place between October 1st and March 7th at Critical Software's facilities in Porto, Portugal.

The project's theme was the development of a converter, part of an Engineering Data Analysis system, which would transform binary STDF (Standard Test Data Format) files into editable XML (Extended Markup Language) based EXF (Engineering Data Analysis Exchange Format) files. This process involved a study of the existing platform and the semiconductor manufacturing process, a requirements definition phase, system modelling, a test case specification, the solution development and the consequent system testing.

The main goals of the project were fulfilled and the initially planned activities were carried out successfully. A preliminary version of the developed solution is already available to be used in the company and the project described in this document is currently on open status in Critical Software.

Special thanks

I would like to thank everyone I worked with at Critical Software for all the help and sympathy, in particular to the persons directly involved in this project – Hugo Casimiro and Helder Ferreira –, the EBS DB project team – Pedro Figueiredo, João Nieto, Thiago Brito and Isabel Gomes –, Vânia Castro and Filipe Pinheiro.

Special thanks to Prof.ª Ana Paula Rocha, from FEUP, for supervising the project, and once again to Hugo Casimiro, for being my tutor at Critical Software.

Last but not least, my dearest ones – family and friends – for the constant support.

Index of contents

Index of figures

Index of tables

Acronyms and abbreviations

ANA: analytical
API: Application Programming Interface
ASCII: American Standard Code for Information Interchange
ATDF: ASCII Test Data Format
ATE: Automatic Test Equipment
CMMI: Capability Maturity Model® Integration
CVS: Concurrent Versions System
DA: Data Archiving
DB: Database
DL: Data Loading
EBS: Engineering Base System
EDL: Extraction Definition Language
EFF: Extraction File Format
ESA: European Space Agency
FCT: functional
FEUP: Faculdade de Engenharia da Universidade do Porto
GUI: Graphical User Interface
IBM: International Business Machines Corporation
JM: Job Management
MDA: Manufacturing Data Analysis
MIEIC: Mestrado Integrado em Engenharia Informática e Computação
MIT: Massachusetts Institute of Technology
NASA: National Aeronautics and Space Administration
PL/SQL: Procedural Language/Structured Query Language
Prof.ª: Professora (Female teacher)
SQL: Structured Query Language
STDF: Standard Test Data Format
UML: Unified Modeling Language
WP: work package
XE: Extraction Engine
XML: Extended Markup Language
XTI: Extraction Tool for Infineon

1 Introduction

1.1 The company Critical Software, SA

Critical Software provides solutions, services and technologies for mission and business critical information systems [1], supporting customers across diverse markets including aerospace, defence, finance, energy, government, manufacturing and telecom.

The company was founded in 1998 and employs more than 200 people, in offices at Coimbra, Lisbon, and Porto (Portugal), San Jose (California), Southampton (United Kingdom) and Bucharest (Romania) [1].

Critical operates with a CMMI® (Capability Maturity Model® Integration) Level 3 certified set of processes across all disciplines of Management and System Engineering and was ranked as one of the fastest growing European Companies in the "Europe 500 Scoreboard" published annually by BusinessWeek [1].

The company has developed strong expertise in several areas of competence, such as:

- Enterprise Application Integration and Databases;
- Dependability and Embedded Software;
- Command and Control;
- Networking Software;
- Software Product Assurance [1].

Some of the products already developed and commercialised by Critical include Xception™ and WMPI™, which are used by diverse companies around the world, mainly in the space, defence, oil and gas sectors.

Critical Software has a high valuable collection of customers, having collaborated with major entities such as NASA (National Aeronautics and Space Administration), Infineon Technologies, Qimonda, MIT (Massachusetts Institute of Technology), Vodafone, Siemens, IBM (International Business Machines Corporation), European Commission, ESA (European Space Agency), among many others.

1.2 The project "STDF2EXF Converter Development" at Critical Software, SA

The project "STDF2EXF Converter Development" took place between October 1st and March 7th at Critical Software's facilities in Porto, Portugal. Aimed to be done by an engineer trainee, in a period of approximately 20 weeks, this project involved designing and implementing a new component in the Engineering Base System (EBS), a system maintained by Critical Software and used by Infineon Technologies within the semiconductor Manufacturing Data Analysis context.

The project came to life with the objective of optimizing existing work tasks executed by some of the Engineering Base System's development teams at Critical Software, as well of presenting a new and possibly rather advantageous solution for one of its modules – the EBS DB (Engineering Base System Database) – to the system's client, Infineon Technologies.

The component to be developed – a STDF2EXF Converter, an application built in C++ that should transform binary files in the industry standard STDF format into Infineon Technologies' proprietary XML based EXF format – would come as a substitute of an existing component of EBS DB – the STDF Loader – introducing into the system more homogeneity, easier file handling and possibly higher maintenance capabilities and better performance. Added to such purposes, this converter should as well be used apart from an EBS DB system installation – working in standalone mode – with the goal of quickly and easily editing binary files in the STDF format. This came as an effort to fight the urge the system's developers had to view or edit information "on the go" of STDF files which are not editable and need to be effectively converted into text files in a short amount of time.

The project milestones comprised gaining business know-how on the semiconductor manufacturing process and the Engineering Base System, gathering and specifying the application requirements and test cases, building a functional vertical prototype, testing and validating the system, and reporting the work done.

The project was successful and the STDF2EXF Converter is already being used in its standalone mode at Critical Software. High priority tasks were performed successfully and low priority tasks were completed in a satisfactory level. The solution was properly documented, requirements were gathered, test cases were specified and the developed prototype was tested. The project status is open.

1.3 The EBS system

The Engineering Base System (EBS) is a Manufacturing Data Analysis (MDA) platform which gets, stores, processes and extracts wafer and chip production data from the test machines of Infineon Technologies' fabric units.

Infineon Technologies is a multinational German company with offices worldwide which offers semiconductors and system solutions for automotive and industrial electronics, chip cards and security, as well as applications in communications [2]. The company's test machines – known as testers – generate wafer and chip production data which is loaded into EBS and stored in a global database, further accessed by experts for analysis and other purposes.

The following pages introduce this system in some detail and give an insight of the semiconductor manufacturing process, two topics which are directly connected with the project's scope.

The subchapters Data collection, EBS and Engineering Data Analysis, together with EBS structure, describe EBS based on what was presented in [5].

Semiconductor production

The chips and integrated circuits present in our everyday electrical and electronic devices are manufactured in a process known as semiconductor device fabrication [3]. It is a multiple step sequence of photographic and chemical processing phases during which electronic circuits are gradually created on a wafer made of pure semiconducting material [3]. Silicon is the most commonly used semiconductor material today, along with various compound semiconductors [3].

The necessary steps to produce a semiconductor integrated circuit can be grouped in two areas:

- Frontend processing;
- Backend processing.

As described in [4], the former occurs in sites known as fabs, where wafers, grouped in lots, are engineered and built, by incorporating transistors directly on the wafer silicon. A fab is one of the most complex industrial facilities to be found anywhere. It has the cleanest environment in the world – many times cleaner than the best hospital operating theatre. The process of wafer fabrication is a series of 16-24 loops, each putting down a layer on the device. At each stage, various inspections and measurements are performed to monitor the process and equipment. Supporting the entire process is a complex infrastructure of materials supply, waste treatment, support, logistics and automation. Once the various semiconductor devices have been created they are interconnected with metal wires to form the desired electrical circuits, connecting the different devices.

Continuing what is mentioned in [4], the backend processing incorporates phases such as testing, assembly and packaging, where the finished wafer is split up into individual dies (chips) that are then assembled into packages, which can be handled in the final applications (Figure 1). A full functional electrical test is performed at both wafer and package level to ensure the outgoing quality of a set of good, usable integrated circuits.

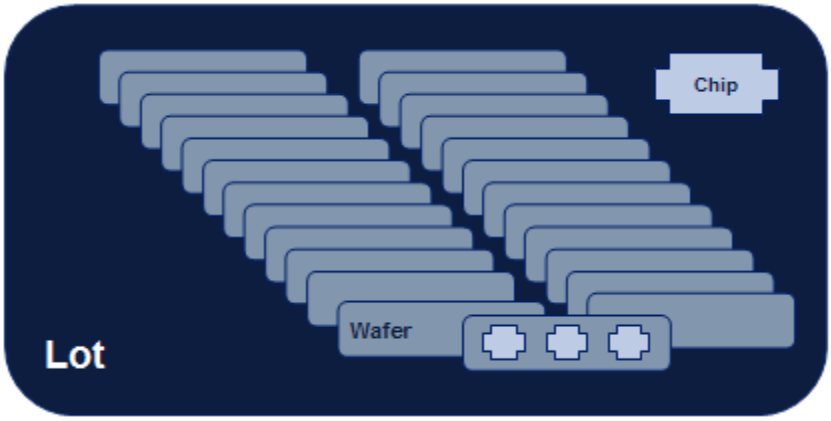

Figure 1: Decomposition of a lot into wafers and chips

Data collection

The data collection by EBS can occur on several levels, since test systems can provide data on a sub-die, die (chip), wafer and lot level. Such data can include information regarding the frontend process – wafer creation, process and testing – as well as the backend process – probe, pre-assembly, assembly and module/component testing. The main data processed by EBS is as follows:

- Wafer processing data, such as film thickness, sheet resistance, particle and defect locations;
- Wafer, component and module electrical test data;
- Lot and wafer tracking data;

- Process slit information.

EBS and Engineering Data Analysis

EBS comprises a set of tools that allow one to supervise chip manufacturing processes, investigating their development and detecting possible errors. The system allows one to:

- Generate diverse types of reports, such as trend charts (bad lots identification), lot reports and detailed wafer reports (Figure 2);
- Analyse wafer maps (Figure 3), which provide a good perspective of the different wafer zones, in order to detect flaws or possible lot defects;
- Compare information between the stored data at EBS and data warehouses;
- Understand the relation between the analysed manufacturing data and the process history, lot history and lot attributes information;
- Define chip, wafer or lot customized extractions;
- Schedule extractions and reports;
- Gather statistics and make use of data mining and recognizing patterns.

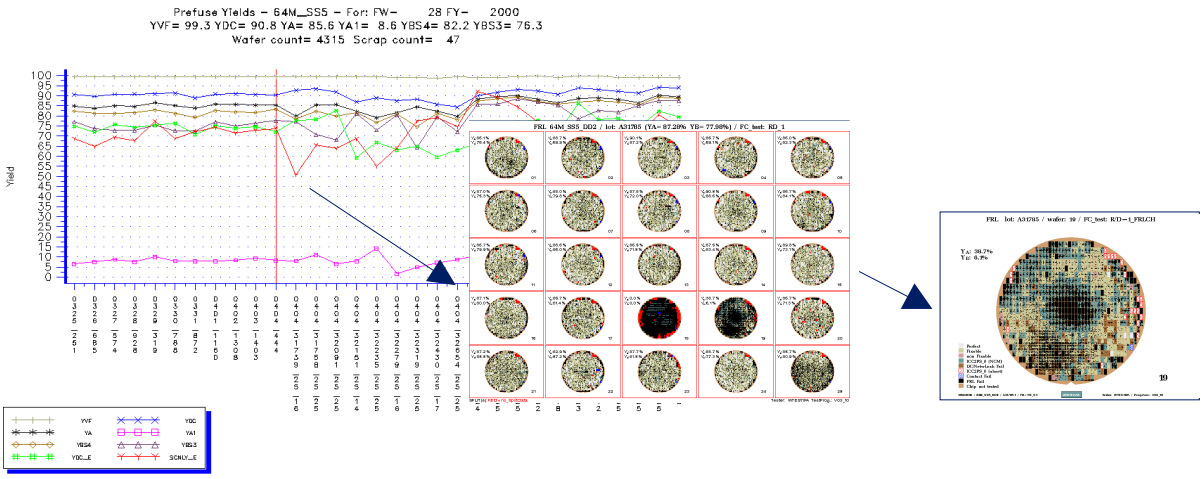

Figure 2: Examples of a trend chart, lot and wafer reports

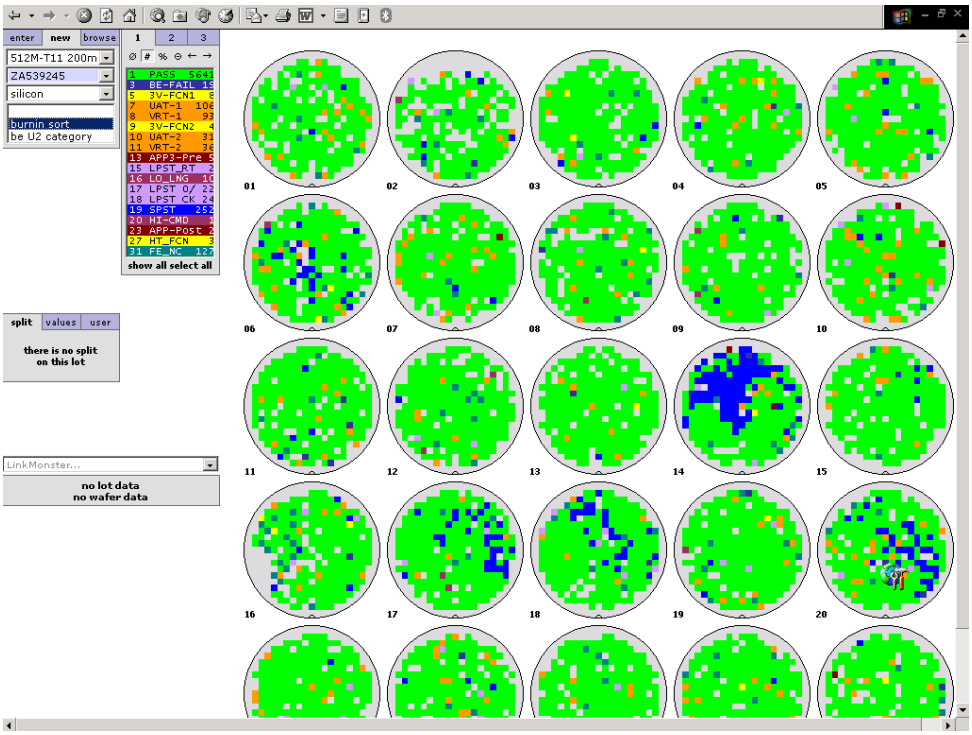

Figure 3: A wafer map example

The usage of such an Engineering Data Analysis tool helps one assure better quality on chip manufacturing by properly identifying lot problems, correlating such problems with their causes and initiating corrective actions.

EBS structure

The EBS system runs on a HP-UX (Hewlett-Packard's Unix based operating system) platform and is divided into different modules, each of them with a unique purpose (Figure 4):

- **Database (EBS DB)** – a data model to store memory and logic data from the frontend and backend processing;
- **Data Loading (EBS DB DL)** – extracts, transforms and loads data into EBS, handling over 50 different data types and executing complex normalization, validation, consolidation and data aggregation operations;
- Extraction Engine (EBS XE) – extraction layer between the database and the user or other applications, retrieving extraction requests in EDL (Extraction Definition Language), translating such information into real SQL (Structured Query Language) statements and generating an EFF (Extraction File Format) file with the results;
- Extraction Tool for Infineon (EBS XTI) – allows the user to define an extraction in EDL;
- Data Archiving (EBS DA) – information archiving and de-archiving according to legal requirements and customer needs.

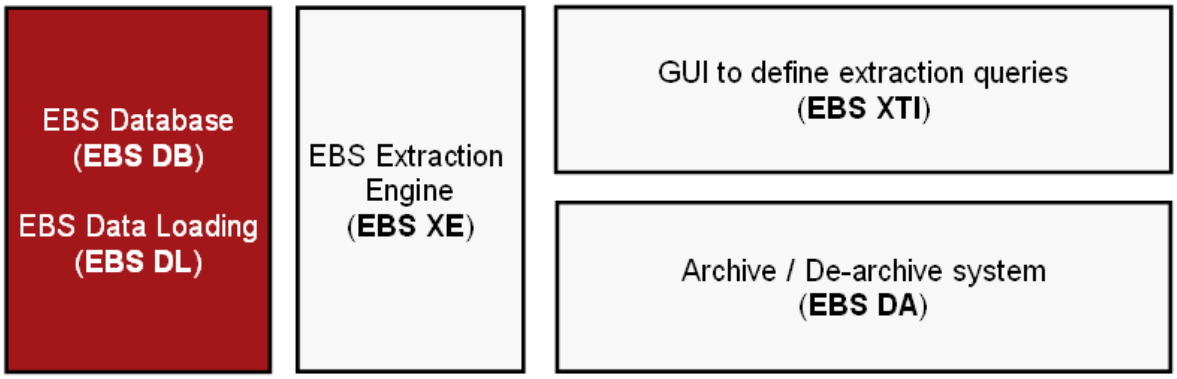

Figure 4: The EBS modules

As highlighted in the text and picture above, the EBS Database and Data Loading modules are those which are directly related to the work developed in this project, therefore being introduced in some detail in the following pages.

Work packages

Every unit that is moved and transformed in EBS is called a work package (WP). It is a folder that contains one or more raw files[1] of the same format inside, e.g., STDF, CP, 2DPATREC, APRC (Figure 5).

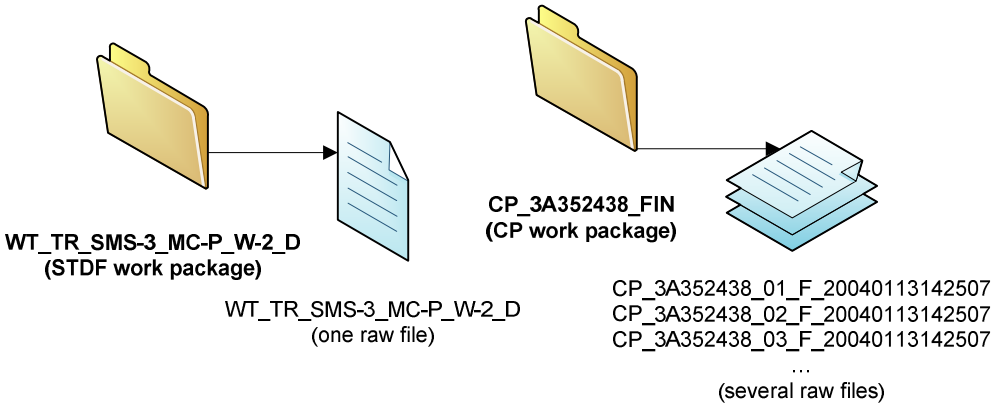

Figure 5: Example of work packages

Work package files contain several records, each with a different meaning or purpose, with information that will be analysed, validated and loaded into the system based on its record type.

Work packages can be categorized in two distinct ways:

- **Master data** or **fact data** work packages;
- **Binary** or **text** based work packages.

[1] The designation of raw files comes from the fact that such files contain raw, untreated information that is extracted directly from test machines – testers – and can be further processed and validated by Engineering Data Analysis tools, such as EBS. Raw files might have different formats, such as STDF or other file formats handled by EBS, such as CP, 2DPATREC, among others.

Master data work packages are usually related to fact data work packages, since their information can be linked with a significant meaning. The former usually include information that the latter require in order to be processed by EBS.

In another perspective, text or ASCII (American Standard Code for Information Interchange) data – 2DPATREC, APRC, CP, among other formats handled by EBS – is easily readable by the human being and editable with any text processing tool, while binary data – STDF format –, on the contrary, is encoded in a way that does not allow one to understand it without decoding it first to text data. EBS supports and processes the types described above in different ways as detailed in the next pages.

EBS Database and Data Loading

The EBS Database module is an aggregation of three different databases, as follows:

- Repository database (EBS_REPO) – stores EBS configurations and supports the system functioning.
- Stage database (EBS_STAGE) – temporary database, which contains information extracted from work packages' raw files.
- Analytical database (EBS_ANA) – final database, where information is saved for further access; this information is usually stored during one year for lot and wafer data and six months for chip data.

EBS Database is supported by EBS Data Loading, which is the process of getting, processing and loading into the analytical database chip and wafer data from Infineon's testers. Such process is executed differently depending if the work packages contain text or binary data. The differences can be seen in Figure 6 and Figure 7.

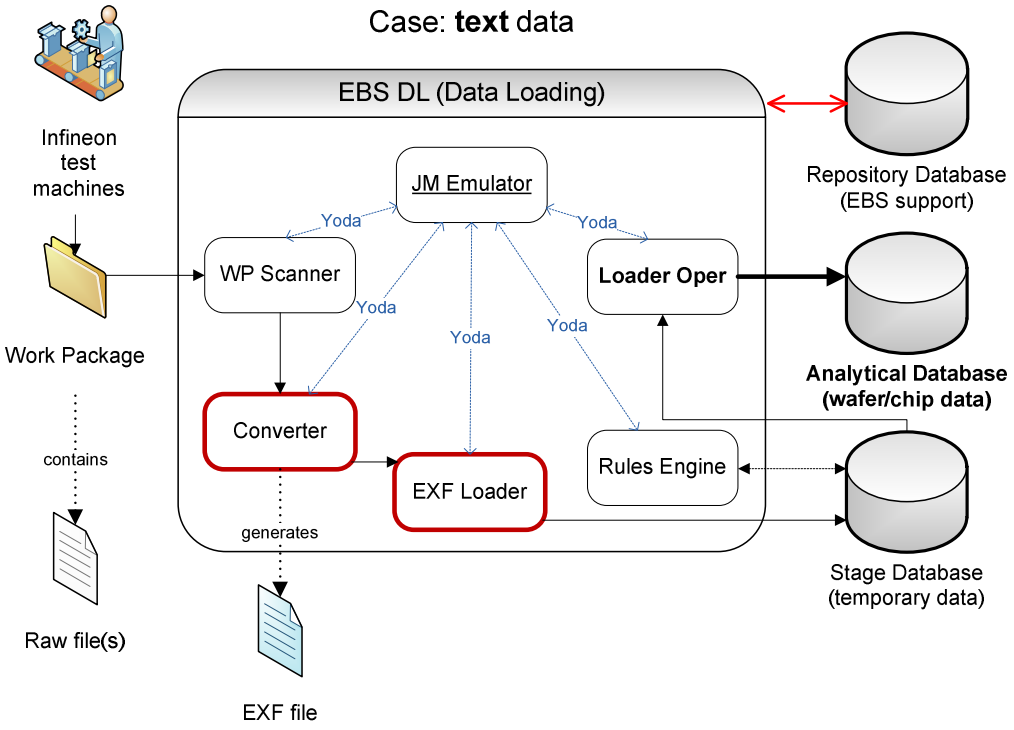

Figure 6: Data loading for text work packages

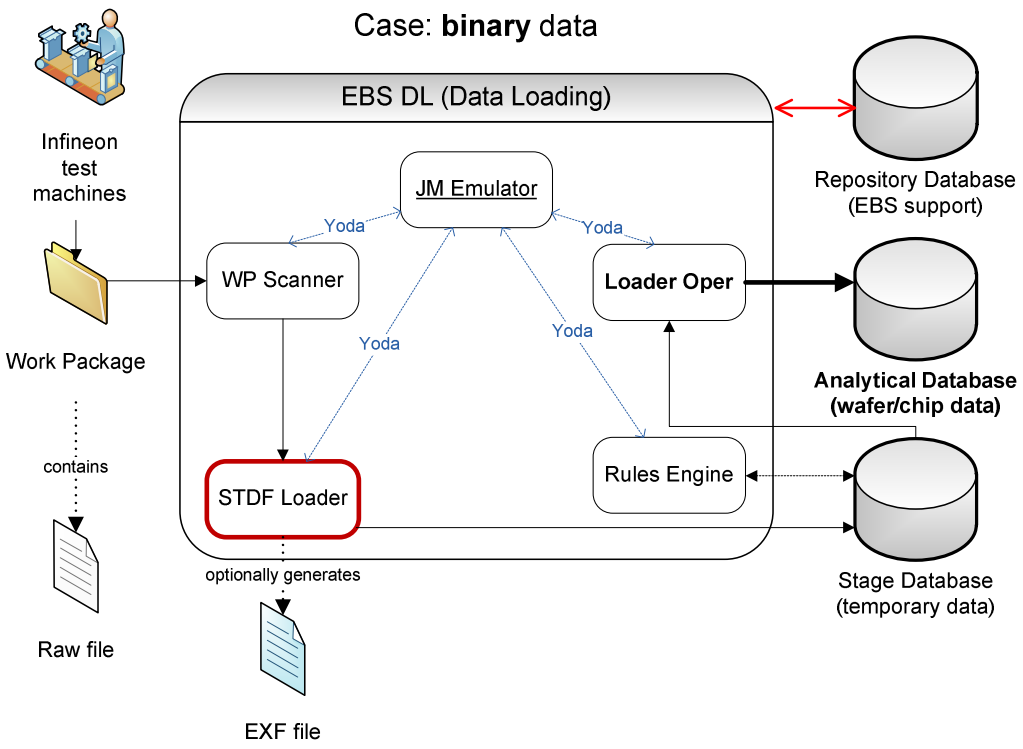

Figure 7: Data loading for binary work packages

The major components that compose the EBS Data Loading module are described as follows:

- JM (Job Management) Emulator – the main process of EBS Data Loading which controls all the other components, initializing and terminating them when needed, through the exchange of YODA[2] (Your Own Data Adapter) messages.
- WP Scanner – scans the system's input data folders and detects the presence of new work packages waiting to be processed; it analyzes the raw files contained in each work package, detects their file types and formats, and registers the work package in the repository database; moves the work package to the appropriate converter for further processing or, if the work package is not valid, moves it to an error folder.
- Converters (text data only) – convert the raw data files included in each work package into EXF files (XML based), which then become ready to be loaded into the staging area (see EXF Loader); each raw file type has its own converter, e.g., 2DPATREC2EXF Converter for 2DPATREC files, APRC2EXF Converter for APRC files.
- EXF Loader (text data only) – loads the EXF files generated from the converters into a stage database – a temporary database where data is analysed and normalized before being moved to a final database.
- STDF Loader (binary data only) – converts and loads STDF files into the stage database, working both as a converter and a loader specifically for such files – the

[2] YODA is an application integration framework developed by Infineon Technologies which provides functionalities for communication, security and configuration of applications that might be running on different operating systems or using different technologies [6]. EBS uses the YODA framework and its components communicate through YODA messages.

main goal of the project is to transform this component into a new converter and migrate its loading logic into the EXF Loader.

- Rules Engine – performs complex operations of analysis, validation, normalization and aggregation on the data loaded into the stage database, in order to prepare it to be loaded into the final database – the analytical database.
- Loader Oper – loads the information from the stage database into the analytical database.

1.4 The STDF format

The Standard Test Data Format (STDF) is a proprietary file format for semiconductor test information originally developed by Teradyne, now widely used throughout the semiconductor industry [7]. It is commonly produced by Automatic Test Equipment (ATE) platforms [7] from many major companies. STDF is a binary format and its extraction to ASCII text is non trivial as it involves a detailed comprehension of the STDF specification, the 2007 version 4 specification being over 100 pages in length [7].

The STDF format comes as an effort to define a common ground that allows testers, database management systems and data analysis software to store and communicate test data in a form that is useful, general, and flexible [8].

As described in [9], using a single format such as STDF gives one many advantages for lining up the process of creating, storing and handling measurement data, namely:

- Data of different tester types can be compared to see if tests are correlating;
- The STDF file format is very compact and saves storage space;
- There are tester types which directly generate this STDF file format.

EBS supports the STDF format, processing and storing such files in the analytical database. This is done, however, in a different way than for other data types, since STDF is a binary format, opposing to the other formats handled by EBS, which are text based. This lead to the introduction of a specific component into the EBS Data Loading module to process STDF files. This component, the STDF Loader, converts raw STDF files into memory and loads their information into the stage database, not following the mechanism used to process text files, in which each file passes through a specific converter and is loaded by the EXF Loader into the stage database. Such differences are the basis for the problem that boosted this project, which will be analysed in more detail in Chapter 2.

1.5 Report's structure

This document is specifically organised to fulfil the goal of describing in detail the different phases of the project and the work that has been done.

Chapter 1, Introduction, gives an overview of the project and its context, namely the EBS system, the semiconductor manufacturing process and the STDF format.

Chapter 2, Problem analysis, describes the problem, the conditions to carry it out, the project planning and its delimitation.

Chapter 3, Technological review, presents the EXF format, the position of the STDF format in the semiconductor industry, the existing solutions for similar problems and the languages that were used during the prototype development.

Chapter 4, Specification of the new solution, comprises the different models that were designed prior to the prototyping stage, together with the gathered requirements, specified use cases and test cases.

Chapter 5, Prototype development, introduces the developed prototype, showing how it was built, the choices that were made throughout its development and the difficulties found.

Chapter 6, Project concretisation, reviews the work that was done, by analysing the system tests and the obtained results, and by comparing the executed tasks with the initial project plan.

Chapter 7, Conclusions and future work perspective, includes an overall summary of the previous chapters, the conclusions drawn throughout and at the end of the project, and demonstrates how the executed work can be carried on in the future.

A project log is included as annex in the end of the document.

The documents Software Requirements Specification [11], Test Case Specification [19], System Test Report [20] and Regression Testing Specification [21], created by the author, and the documents Standard Test Data Format (STDF) Specification Version 4 [8] and EXF Schema Specification [12], are included as annexes to this document in electronic format.

2 Problem analysis

The problem analysis reported in this document is not part of the project's work since the problem and its possible solutions had already been taken into consideration by the project manager and its main stakeholders, who had already defined its goals in advance.

Below is shown a table containing an overview of the project's problem, its possible solutions and their advantages. Further discussion is based on what was described in [11].

Problems	Solutions
STDF files are processed differently than other file types - Extra component: STDF Loader - Heterogeneous system	**Process binary raw files similarly to text raw files** - Develop a new standardized converter that transforms binary raw STDF files into XML text based EXF files - Eliminate the STDF Loader and integrate the new developed converter into the existing system, making it in this way homogeneous
It is difficult to quickly view and edit a STDF raw file - Hard to edit binary data - The EBS system needs to be installed and compiled in order to achieve this	**Introduce functionality that allows one to quickly decode or edit a STDF raw file** - Generate easily editable XML files from STDF raw data - Perform this independently, i.e., running apart from the EBS system

Table 1: Problem description overview

The EBS system processes and loads into a database text and binary raw data files, which contain chip and wafer test data, created by Infineon's test machines.

The EBS system currently processes text raw data files, e.g., 2DPATREC, ChipXRef or FAB300EDC, in a different way than binary raw data files, e.g., STDF. While the former are firstly processed by a specific converter and then loaded into the database with a standard loader component – EXF Loader –, the latter are converted and loaded directly into the database using a specific loader – STDF Loader.

Figure 8 shows the flow of STDF data in the current system.

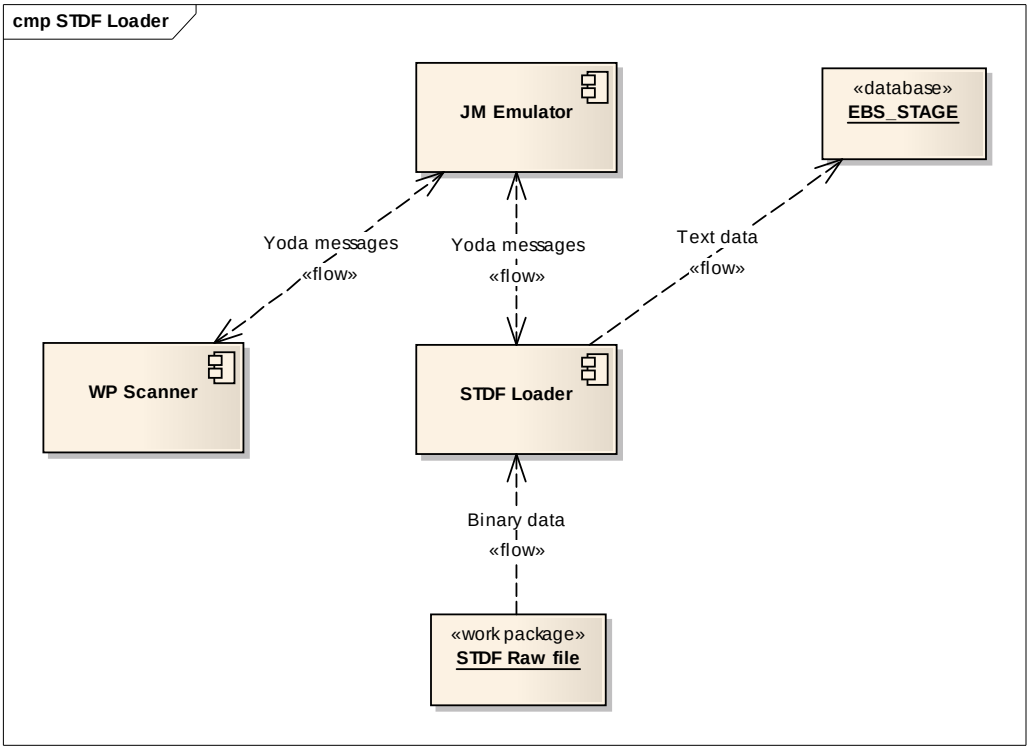

Figure 8: The current processing of STDF files

As seen above, STDF binary raw data is converted into text data and loaded into the EBS_STAGE database using a single component – the STDF Loader. This component converts the input binary data in memory and loads it afterwards into the database, not generating any files during this process.

This approach has two issues, as follows:

- The STDF data loading process is not achieved in a standard way: all the other data type files are converted into XML based EXF files using a specific converter, e.g., 2DPATREC2EXF Converter for 2DPATREC files or APRC2EXF Converter for APRC files, and then loaded into the database with the EXF Loader component.
- STDF files contain binary data, which is difficult to interpret, and quite often the system administrators need this information to be quickly accessible in a XML (EXF) file. This is not possible in the current system, since the STDF Loader does not generate output EXF files from the conversion, transforming instead the data in memory before its loading into the database.

Figure 9 shows a model for a new solution proposed in this thesis, in which the system shall process all data types in a homogeneous way, with the introduction of a STDF2EXF Converter for the conversion of STDF files.

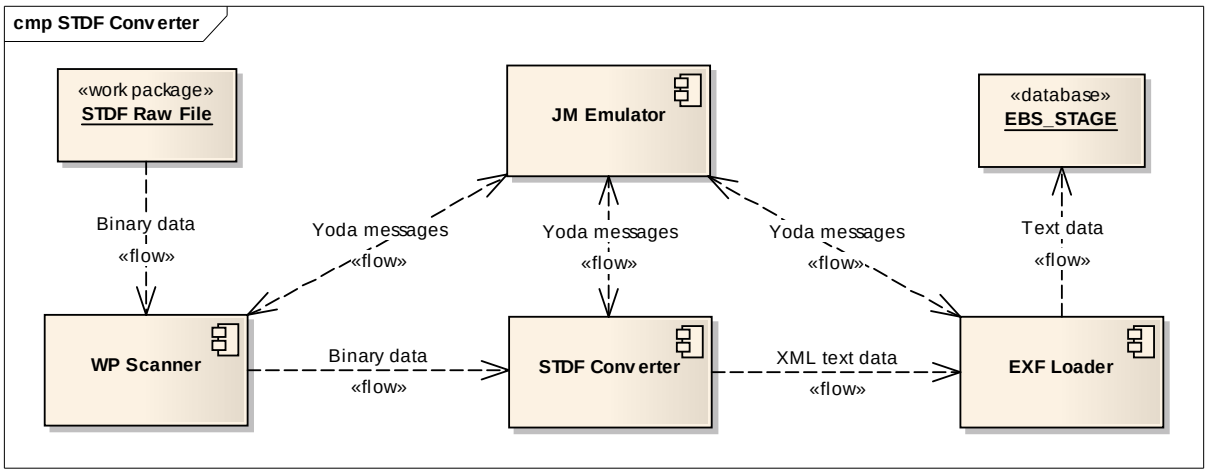

Figure 9: A prototype for the future processing of STDF files

A new STDF2EXF Converter should be added to the EBS Data Loading process, replacing the existing STDF Loader and incorporating its main conversion logic. The converter should process binary raw STDF files and generate EXF files as output, which should be loaded into the EBS database by the EXF Loader component.

Once the STDF2EXF Converter is implemented, the STDF loading logic of the STDF Loader should be migrated and integrated into the EXF Loader, to allow such files to be loaded into the staging database successfully and as expected.

After the implementation of the converter and its full integration with EBS DB it should be possible to evaluate the processing of STDF files, comparing the performances of the new and old systems.

2.1 Persons and parties involved

This was an individual project, planned by a project manager from Critical Software, supervised by a designated tutor at the company and a selected teacher at FEUP. Table 2 shows the persons and parties involved in the project.

Persons and parties directly involved	Roles
Daniel Silva	Project development, trainee
Helder Ferreira	Project manager
Hugo Casimiro	Tutor (EBS DB team)
Prof.ª Ana Paula Rocha	Supervisor from FEUP

Table 2: Persons and parties involved in the project

2.2 Resources, equipment and knowledge

The paragraphs below describe some pertinent points regarding the work to be done.

Would we have the knowledge needed?

EBS is a complex system that one needs to be introduced to through a series of formation sessions and documentation reading. The semiconductor manufacturing process would also

need to be a subject of study, since many of the concepts approached in this project are related to this matter. Nevertheless, no previous knowledge about this subject was required in order to being able to take the project ahead.

Although the converter's programming would be made using C++, a language which was not studied by the trainee prior to this project, this matter was not seen as a barrier to the work to develop, since C++ has an object oriented nature that is common to the languages the author had experience in. The trainee was already familiar with the other languages and environments used in this project such as XML, SQL and Unix.

Would we have the time needed?

The project was planned in a way that would allow the work to be distributed by balanced time windows, allowing the necessary time to carry on the different tasks. The time initially allocated for the project was somehow limited, in a way that it would allow the implementation of the STDF2EXF converter, not leaving however any open time window for this component's integration with the EBS system.

Would we have the material needed?

The company was prepared with all the physical resources and equipment needed in order to respond to the foreseen demands of the project.

2.3 Plan and procedure

The project was planned in accordance with the amount of work to be done, between the project manager, tutor and trainee, having been approved by the supervisor from FEUP, Prof.ª Ana Paula Rocha. All tasks were executed by the trainee, reviewed and approved by the project tutor, Hugo Casimiro, in accordance to Critical Software's existing Quality Assurance procedures.

The project plan was updated throughout the project and significantly changed in mid January. The two plan versions are detailed in the sections below.

Initial project plan – October 2007

The initial Gantt chart with the project tasks, milestones and deliverables is shown in Figure 10.

ID	WBS	Task Name	Duration	Start	Finish
1	DS	**Daniel Internship PLN**	**103,5 days**	**Mon 01-10-07**	**Thu 28-02-08**
2	**WP-1000**	**Business Know-How**	**9,5 days**	**Mon 01-10-07**	**Mon 15-10-07**
3	WP-1100	EBS Data Loading	1 day	Mon 01-10-07	Mon 01-10-07
4	WP-1200	EBS Data Model	2 days	Tue 02-10-07	Wed 03-10-07
5	WP-1300	Converters Framework	2 days	Thu 04-10-07	Mon 08-10-07
6	WP-1400	STDF Format	0,5 days	Tue 09-10-07	Tue 09-10-07
7	WP-1500	EXFLoader	2 days	Tue 09-10-07	Thu 11-10-07
8	WP-1600	STDFLoader	2 days	Thu 11-10-07	Mon 15-10-07
9	**WP-2000**	**Requirements Engineering Phase**	**20 days**	**Mon 15-10-07**	**Tue 13-11-07**
10	WP-2100	Software Requirement Specification	10 days	Mon 15-10-07	Mon 29-10-07
11	WP-2200	Test Case Specification	10 days	Mon 29-10-07	Tue 13-11-07
12	**WP-3000**	**Phase 1**	**47 days**	**Tue 13-11-07**	**Mon 21-01-08**
13	**WP-3100**	**Design Engineering Phase**	**30 days**	**Tue 13-11-07**	**Wed 26-12-07**
14	WP-3110	Code implementation	30 days	Tue 13-11-07	Wed 26-12-07
15	**WP-3200**	**Validation Phase**	**7 days**	**Wed 26-12-07**	**Mon 07-01-08**
16	WP-3210	System Testing	5 days	Wed 26-12-07	Thu 03-01-08
17	WP-3220	Update documentation	2 days	Thu 03-01-08	Mon 07-01-08
18	**WP-3300**	**Internship Documentation**	**10 days**	**Mon 07-01-08**	**Mon 21-01-08**
19	WP-3310	Write documentation	10 days	Mon 07-01-08	Mon 21-01-08
20	**WP-4000**	**Phase 2**	**22 days**	**Mon 21-01-08**	**Thu 21-02-08**
21	**WP-4100**	**Design Engineering Phase**	**15 days**	**Mon 21-01-08**	**Tue 12-02-08**
22	WP-4110	Code implementation	15 days	Mon 21-01-08	Tue 12-02-08
23	**WP-4200**	**Validation Phase**	**7 days**	**Tue 12-02-08**	**Thu 21-02-08**
24	WP-4210	System Testing	5 days	Tue 12-02-08	Tue 19-02-08
25	WP-4220	Update documentation	2 days	Tue 19-02-08	Thu 21-02-08
26	**WP-5000**	**Performance Study**	**5 days**	**Thu 21-02-08**	**Thu 28-02-08**
27	WP-5100	Run Regression Tests	3 days	Thu 21-02-08	Tue 26-02-08
28	WP-5200	Compare Results	1 day	Tue 26-02-08	Wed 27-02-08
29	WP-5300	Write Performance Report	1 day	Wed 27-02-08	Thu 28-02-08
30					
31	ML	**Milestones**	**103,5 days**	**Mon 01-10-07**	**Thu 28-02-08**
32	ML-01	KOM	0 days	Mon 01-10-07	Mon 01-10-07
33	ML-02	Business Know-How Complete	0 days	Mon 15-10-07	Mon 15-10-07
34	ML-03	Requirements ready	0 days	Tue 13-11-07	Tue 13-11-07
35	ML-04	Source Code is implemented	0 days	Wed 26-12-07	Wed 26-12-07
36	ML-05	Source Code is Validated	0 days	Mon 07-01-08	Mon 07-01-08
37	ML-06	Internship Report is done	0 days	Mon 21-01-08	Mon 21-01-08
38	ML-07	EXFLoader is implemented	0 days	Tue 12-02-08	Tue 12-02-08
39	ML-08	EXFLoader is tested	0 days	Thu 21-02-08	Thu 21-02-08
40	ML-09	Performance conclusions are available	0 days	Thu 28-02-08	Thu 28-02-08
41					
42	DL	**Deliverables**	**74 days**	**Tue 13-11-07**	**Thu 28-02-08**
43	DL-01	Software Requirement Specification	0 days	Tue 13-11-07	Tue 13-11-07
44	DL-02	Test Case Specification	0 days	Tue 13-11-07	Tue 13-11-07
45	DL-03	Test Log Report (Phase 1)	0 days	Mon 07-01-08	Mon 07-01-08
46	DL-04	Source Code (Phase 1)	0 days	Mon 07-01-08	Mon 07-01-08
47	DL-05	Internship Report	0 days	Mon 21-01-08	Mon 21-01-08
48	DL-06	Test Log Report (Phase 2)	0 days	Thu 21-02-08	Thu 21-02-08
49	DL-07	Source Code (Phase 2)	0 days	Thu 21-02-08	Thu 21-02-08
50	DL-08	Performance Report	0 days	Thu 28-02-08	Thu 28-02-08

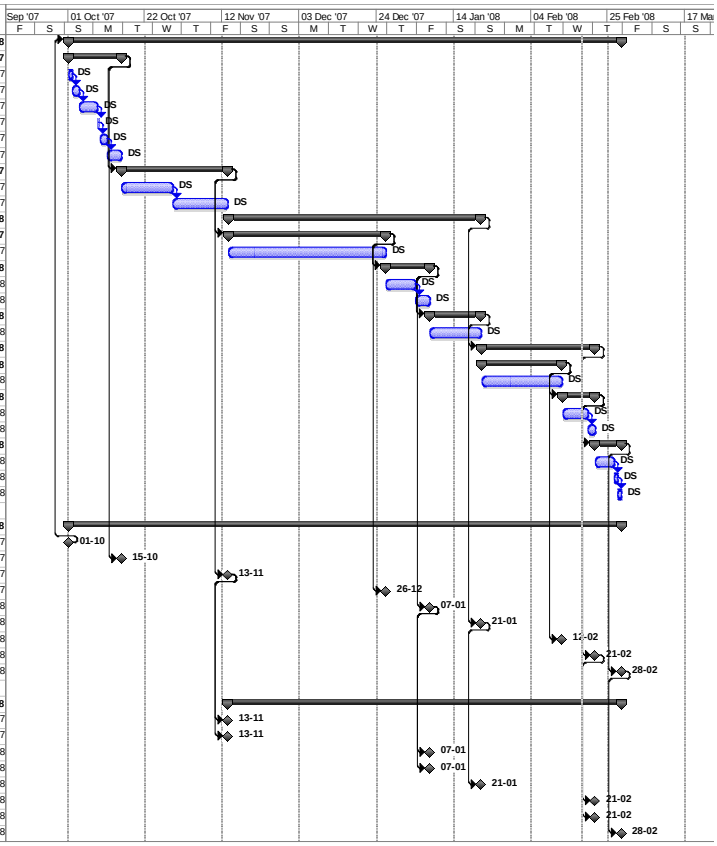

Figure 10: Initial project plan

The first phase of the project had the goal of obtaining the business know-how and studying the platform on which the work would focus. This involved getting familiar with the semiconductor manufacturing process, the EBS system, the STDF and EXF formats, and the architecture and design of EBS Data Loading's main features, i.e., STDF Loader, EXF Loader, EBS data model and the converters framework.

Once a solid knowledge base had been built, the requirements engineering phase should follow, having as deliverables a software requirements specification and a test case specification, as the basis of the work that would follow.

As one can see in the figure above, the project was divided into two phases after the Requirements Engineering Phase. According to MIEIC's regulations, the project should last 20 weeks, an amount of time which would not be enough to complete the project defined in accordance to Critical Software's interests and the six months contract made with the trainee. Given the circumstances, it was decided in a meeting prior to the project start that only a portion of the project would be presented to FEUP and the remaining portion would be done as part of the internship at Critical Software. The plan was, thus, divided as follows:

- Phase 1, included in the protocol defined between Critical Software and FEUP, which did not comprise more than 20 weeks, according to FEUP's regulations.
- Phase 2, exclusive to Critical Software, not presented to FEUP.

In Phase 1, a prototype for the STDF2EXF Converter to be used apart from EBS, working in standalone mode, should be built. Both Phase 1 and Phase 2 should be followed by a validation period in which the system would be tested and approved. In the end of this first phase, two weeks would be allocated to allow the writing of this document.

Phase 2 would involve integrating the created STDF2EXF Converter with the existing EXF Loader, in order to complete the workflow started with the introduction of this converter, so that STDF files could be fully loaded into the EBS system using this new component.

The last set of tasks, following Phase 2, would include running the system regression tests and writing a performance report. These tests would have the goal of comparing the performances of a system with the STDF Loader against a system with the new STDF2EXF Converter.

Updated project plan – January 2008

The project plan was significantly updated in mid January, due to changes on MIEIC's Project regulations for its initial semester of activity since it has been adopted by the course's direction. The project report delivery deadline was moved from February 29th to April 11th 2008, allowing in this way an extra margin for the project completion, originating a new plan as seen in Figure 11.

ID	WBS	Task Name	Duration	Start	Finish
1	DS	**Daniel Internship PLN**	**105 days**	**Mon 01-10-07**	**Fri 29-02-08**
2	**WP-1000**	**Business Know-How**	**9,5 days**	**Mon 01-10-07**	**Mon 15-10-07**
3	WP-1100	EBS Data Loading	1 day	Mon 01-10-07	Mon 01-10-07
4	WP-1200	EBS Data Model	2 days	Tue 02-10-07	Wed 03-10-07
5	WP-1300	Converters Framework	2 days	Thu 04-10-07	Mon 08-10-07
6	WP-1400	STDF Format	0,5 days	Tue 09-10-07	Tue 09-10-07
7	WP-1500	EXFLoader	2 days	Tue 09-10-07	Thu 11-10-07
8	WP-1600	STDFLoader	2 days	Thu 11-10-07	Mon 15-10-07
9	**WP-2000**	**Requirements Engineering Phase**	**20 days**	**Mon 15-10-07**	**Tue 13-11-07**
10	WP-2100	Software Requirement Specification	10 days	Mon 15-10-07	Mon 29-10-07
11	WP-2200	Test Case Specification	10 days	Mon 29-10-07	Tue 13-11-07
12	**WP-3000**	**Phase 1**	**37 days**	**Wed 31-10-07**	**Fri 21-12-07**
13	**WP-3100**	**Design Engineering Phase**	**30 days**	**Wed 31-10-07**	**Wed 12-12-07**
14	WP-3110	Code implementation	30 days	Wed 31-10-07	Wed 12-12-07
15	**WP-3200**	**Validation Phase**	**7 days**	**Thu 13-12-07**	**Fri 21-12-07**
16	WP-3210	System Testing	5 days	Thu 13-12-07	Wed 19-12-07
17	WP-3220	Update documentation	2 days	Thu 20-12-07	Fri 21-12-07
18	**WP-4000**	**Phase 2**	**22 days**	**Wed 09-01-08**	**Fri 08-02-08**
19	**WP-4100**	**Design Engineering Phase**	**15 days**	**Wed 09-01-08**	**Tue 29-01-08**
20	WP-4110	Code implementation	15 days	Wed 09-01-08	Tue 29-01-08
21	**WP-4200**	**Validation Phase**	**7 days**	**Wed 30-01-08**	**Fri 08-02-08**
22	WP-4210	System Testing	5 days	Wed 30-01-08	Wed 06-02-08
23	WP-4220	Update documentation	2 days	Thu 07-02-08	Fri 08-02-08
24	**WP-5000**	**Performance Study**	**5 days**	**Mon 11-02-08**	**Fri 15-02-08**
25	WP-5100	Run Regression Tests	3 days	Mon 11-02-08	Wed 13-02-08
26	WP-5200	Compare Results	1 day	Thu 14-02-08	Thu 14-02-08
27	WP-5300	Write Performance Report	1 day	Fri 15-02-08	Fri 15-02-08
28	**WP-3300**	**Internship Documentation**	**10 days**	**Mon 18-02-08**	**Fri 29-02-08**
29	WP-3310	Write documentation	10 days	Mon 18-02-08	Fri 29-02-08
30					
31	ML	**Milestones**	**105 days**	**Mon 01-10-07**	**Fri 29-02-08**
32	ML-01	KOM	0 days	Mon 01-10-07	Mon 01-10-07
33	ML-02	Business Know -How Complete	0 days	Mon 15-10-07	Mon 15-10-07
34	ML-03	Requirements ready	0 days	Tue 13-11-07	Tue 13-11-07
35	ML-04	Source Code is implemented	0 days	Wed 12-12-07	Wed 12-12-07
36	ML-05	Source Code is Validated	0 days	Fri 21-12-07	Fri 21-12-07
37	ML-06	Internship Report is done	0 days	Fri 29-02-08	Fri 29-02-08
38	ML-07	EXFLoader is implemented	0 days	Tue 29-01-08	Tue 29-01-08
39	ML-08	EXFLoader is tested	0 days	Fri 08-02-08	Fri 08-02-08
40	ML-09	Performance conclusions are available	0 days	Fri 15-02-08	Fri 15-02-08
41	ML-10	Start work on phase 2	0 days	Wed 09-01-08	Wed 09-01-08
42					
43	DL	**Deliverables**	**75,5 days**	**Tue 13-11-07**	**Fri 29-02-08**
44	DL-01	Software Requirement Specification	0 days	Tue 13-11-07	Tue 13-11-07
45	DL-02	Test Case Specification	0 days	Tue 13-11-07	Tue 13-11-07
46	DL-03	Test Log Report (Phase 1)	0 days	Fri 21-12-07	Fri 21-12-07
47	DL-04	Source Code (Phase 1)	0 days	Fri 21-12-07	Fri 21-12-07
48	DL-05	Internship Report	0 days	Fri 29-02-08	Fri 29-02-08
49	DL-06	Test Log Report (Phase 2)	0 days	Fri 08-02-08	Fri 08-02-08
50	DL-07	Source Code (Phase 2)	0 days	Fri 08-02-08	Fri 08-02-08
51	DL-08	Performance Report	0 days	Fri 15-02-08	Fri 15-02-08

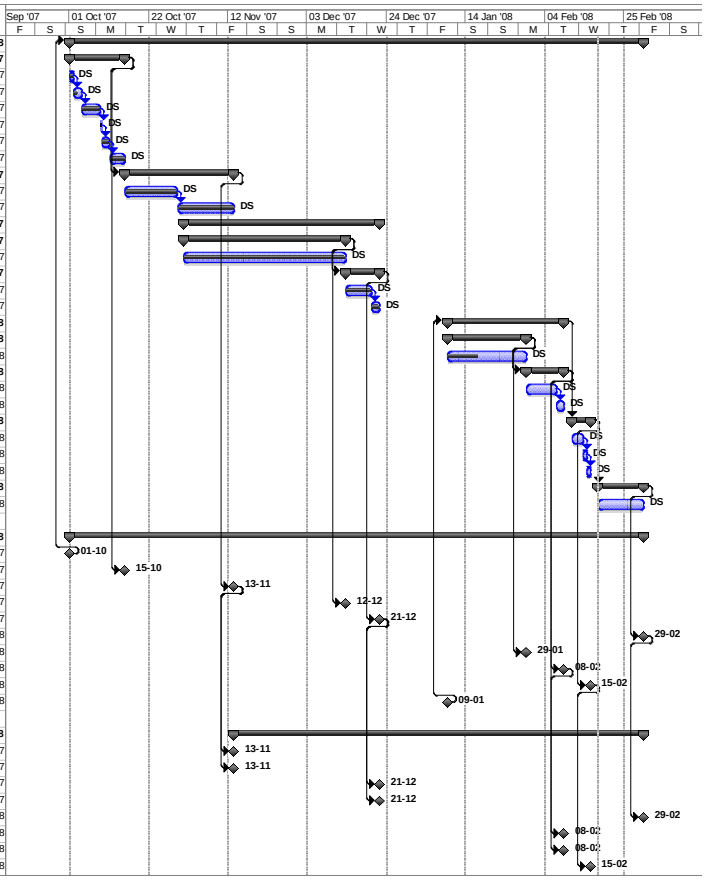

Figure 11: Updated project plan

In the new plan, Phase 1 and Phase 2 were aggregated and the report writing stage was moved so that the entire project could be completed in time of being presented to FEUP and included in this document.

2.4 Requirements and delimitation

This project was delimited based on time constraints and its complexity, with its requirements having been divided by priority.

The following goals were set as being high priority:

- The development of a prototype for the STDF2EXF Converter, which could run in standalone mode, apart from any EBS DB installation;
- The project report writing, together with the converter's documentation, namely Critical Software's Software Requirement Specification and Test Case Specification documents.

The following goals were set as being low priority (not part of the initial plan but included in its second version of January 2008):

- The full integration of the STDF2EXF converter with the EXF Loader and EBS DB;
- A performance study involving the comparison between the new system and the previous one, if time permitted, after the full integration of the STDF2EXF Converter with the system.

3 Technological review

This chapter presents the state of the art surrounding the problem approached in this work. In the next pages an overview of the EXF format is made, as described in [12], followed by an insight into the STDF format and existing associated applications. An explanation of how existing solutions could be reused in this project is made in the next subchapter, followed by an overview of the programming languages used during design and development.

3.1 The EXF format

Engineering Data Analysis Exchange Format (EXF) is Infineon Technologies' exchange format between the company's Manufacturing Data Analysis (MDA) databases and MDA tools. It is an Intermediate Data Format (IDF) into which raw data files are converted in order to be loaded into a database. It is XML based and should be used for all kinds of complex data structures which can not be expressed in a simple two dimensional table. The format is used for general data exchange purposes, such as data extraction, although it was designed with focus on data loading.

File types

There are two file types associated with EXF, being the EXF schema and the instance documents.

The former is a XML Schema document (.xsd) that describes the structure and the syntax of EXF data. Moreover, it includes the constraint definitions, such as primary, foreign and unique keys, enumeration and data types, to ensure the data consistency and some general database relations. This document is the actual EXF format description in XML language and it is used to validate EXF data against the rules and structure specified in the XML Schema.

An EXF instance document contains the actual data expressed in XML syntax. The allowed elements, attributes and structure are defined in the EXF schema, which can be used to validate the EXF data. Validation is an optional process when parsing XML data and requires a capable parser and a reference in the XML root element (for EXF data it is the root element "`EXF`") to the location of the corresponding schema document.

Structure

The XML elements of the EXF file have no hierarchical structure, mainly due to the EXF file size restrictions and the compliance to the EBS data model.

Raw files can be substantially large, and the conversion from raw data types to XML causes an additional overhead on the file size. One of the EXF requirements states that the ratio of uncompressed EXF against raw data files can not exceed 10, as the ratio of compressed EXF against raw data files can not exceed 5. To fulfil this requirement, some compromises were made to the pure XML definition, e.g., content of certain elements consists of comma separated values, which needs to be interpreted by the application program without or with less help from the XML schema metadata.

The EBS data model is designed to support all Engineering Data Analysis data sources and its business processes. Changes in the data model will also lead to adjustments and modifications

on the EXF format specification. The frequency of data model changes should however reduce with the lifetime and stabilization process of the EBS framework.

Figure 12 shows an example of an EXF instance document, based on real data.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Thomas Giegold (Infineon
Technologies/I.S.) -->
<EXF xmlns="http://www.infineon.com/exf" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.infineon.com/exf
../EXF_Schema.xsd" Version="V01.00">
      <TrackingUnit ID="1" TUType="WAFER" CreateDate="2002-08-22T16:20:30"
MotherLotID="A95033" WafNr="15"/>
      <MeasAllowed ID="2" MeasStep="TVM" MeasDataSource="PCM" SumDesc=""/>
      <Measurements ID="3" MeasAllowedID="2" MeasCounter="1" CompSeqNr="0"
TimestampBegin="2002-07-22T00:55:30" TimestampEnd="2002-07-22T01:19:24"
TimestampInsertion="2002-07-22T01:27:27" MeasLotID="A95033.1" MeasCategory="PRODUCTIVE"
SumMeasID="1" TUID="1"/>
      <MeasZoneVals ID="4" ZoneType="NOZONE" Val="NOVAL"/>
      <EnvVals MeasID="5" EnvType="PROBER" ZoneValID="4" Val="PROB92PA"/>
      <EnvVals MeasID="6" EnvType="TESTER" ZoneValID="4" Val="CLAS92PA"/>
      <EnvVals MeasID="7" EnvType="DESIGN_ORIG" ZoneValID="4" Val="256M_D14_DD2"/>
      <EnvVals MeasID="8" EnvType="BASIC_DESIGN" ZoneValID="4" Val="256M-D14"/>
      <EnvVals MeasID="9" EnvType="SUB_DESIGN" ZoneValID="4" Val="DD2"/>
      <ParmDefs ID="10" MeasAllowedID="2" ParmType="ANA" Name="xBV_C32_AD_PW"/>
      <ParmDefs ID="11" MeasAllowedID="2" ParmType="ANA" Name="xCA_C04_AD_PW"/>
      <ParmDefs ID="13" MeasAllowedID="2" ParmType="ANA" Name="DL_N_023"/>
      <MeasPartPos MeasID="3" PosType="RETORIG">
            <Pos X="14" Y="1"/>
            <Pos X="6" Y="5"/>
            <Pos X="26" Y="5"/>
            <Pos X="2" Y="8"/>
            <Pos X="14" Y="8"/>
            <Pos X="18" Y="10"/>
            <Pos X="10" Y="11"/>
            <Pos X="6" Y="12"/>
            <Pos X="26" Y="12"/>
            <Pos X="14" Y="16"/>
      </MeasPartPos>
      <MeasPartVals MeasID="1" ParmID="1">
            <NumVal>-1.144900E+01 -1.119500E+01 -1.102400E+01 -1.098800E+01 -1.123500E+01
-1.116200E+01 -1.109900E+01 -1.091100E+01 -1.095200E+01 -1.085800E+01</NumVal>
      </MeasPartVals>
      <MeasPartVals MeasID="1" ParmID="2">
            <NumVal>+1.092825E-11 +1.118575E-11 +1.151915E-11 +1.132575E-11 +1.109375E-11
+1.107695E-11 +1.120295E-11 +1.130855E-11 +1.170305E-11 +1.171565E-11</NumVal>
      </MeasPartVals>
      <MeasPartVals MeasID="1" ParmID="3">
            <NumVal>+1.946534E+01 +2.049802E-02  +3.633683E+01 +2.157063E+01 +1.783775E+01
+1.383182E+01 +1.781600E+01 +2.094952E+01 +3.505575E+00 +2.116847E+01</NumVal>
      </MeasPartVals>
</EXF>
```

Figure 12: EXF instance document example

A superficial analysis of the EXF file structure is described in 5.1 Prototype engineering.

3.2 STDF and the ATE industry

The STDF file format specification document was created in the 80's, by Teradyne [10]. STDF was developed with the intent of being a simple, flexible, portable data format to which existing data files and formats could be easily and economically converted [8]. Teradyne aimed to cover the existing lack of test result data compatibility between test systems of different manufacturers, and sometimes within the product lines of a single manufacturer [8], by defining a file format used to store test results and making this file format a standard so that the whole industry could benefit from it. By doing so, ATE users and ATE manufacturers

were able to support a common database format and different brands of testers started working with a unified statistical package.

As stated in [8], "STDF is flexible enough to meet the needs of the different testers that generate raw test data, the databases that store the data, and the data analysis programs that use the data. The fact that it is a single, coherent standard also facilitates the sharing and communicating of the data among these various components of the complete ATE system. STDF is not an attempt to specify a database architecture for either testers or the centralized database engines. Instead, it is a set of logical record types. Because data items are described in terms of logical record types, the record types can be used as the underlying data abstraction, whether the data resides in a data buffer, resides on a mass storage device, or is being propagated in a network message. It is independent of network or database architecture. Furthermore, the STDF logical record types may be treated as a convenient data object by any of the software, either networking or database, that may be used on a tester or database engine."

With STDF it is possible for a single data formatting program running on a centralized database engine to accept data from a wide range of testers, whether the testers come from one vendor or from different vendors or are custom-built by the ATE user. This data can be exported to the user's in-house network for further analysis in a form that is well documented and thoroughly debugged [8].

The major objectives of the STDF format, as stated in version 4 of its specification, are the following:

- Be capable of storing test data for all semiconductor testers and trimmers;
- Provide a common format for storage and transmission of data;
- Provide a basis for portable data reporting and analysis software;
- Decouple data message format and database format to allow enhancements to either, independently of the other;
- Provide support for optional (missing or invalid) data;
- Provide complete and concise documentation for developers and users;
- Make it easy for customers to write their own reports or reformat data for their own database.

In the ATE industry, STDF is the most commonly used data format for storing measurement data in digital format [9]. The acceptance of this file format as a standard lead to its gradual adoption by several companies that are part of the ATE industry, such as Teradyne, Verigy, LTX, Credence [7] or SZ [10], giving birth to different kinds of both free and commercial STDF converters and applications used in various distinct contexts and purposes.

Some of these applications that approach similar problems as the one studied in this project are mentioned below.

PySTDF

PySTDF, developed by Casey Marshall and released under a GPL license, is a Python module for processing STDF, which provides event-based stream parsing of STDF version 4, along

with indexers that can help one rearrange the data into a more useful tabular form, as well as generate missing summary records or new types of derivative records [13].

Potential applications of PySTDF include:

- Debugging a vendor's STDF implementation;
- Straight conversion to ASCII-readable form – STDF to ATDF (ASCII Test Data Format) converter;
- Repairing STDF files;
- Developing an application that leverages STDF:
    - Conversion to tabular form for statistical analysis tools;
    - Loading data into a relational database [14].

Casey Marshall developed as well a basic STDF viewer named STDF Explorer, which allows one to explore contents of STDF records.

SEDana

SEDana, as described in [15], is a tool from Salland Engineering for statistical analysis of ATE test data, which is able to import, export or merge STDF version 4, CSV and ATDF files, among others.

Test data and analysis results can be displayed in various table formats. A raw table displays the raw (filtered) STDF data. The user can verify the data using this view or use parts of the data by copying it to another application, e.g., Excel. The data in this table is static, but the user is able to edit values in this view.

The application contains a STDF to ATDF converter.

Spry Software

Spry Software, as described in [16], offers a range of tools that handle STDF files.

QuickEdit STDF viewer and editor allow accessing data in STDF files. Parametric results by part, software bin summaries, hardware bin summaries and test configuration data are all visible in a tabular form. The application allows exporting any or all of these types of data in comma separated files, suitable for loading into Excel or any analysis tool. Data can as well be edited and re-saved in STDF format.

QuickChange provides a set of commands which allow one to convert STDF files to CSV, XML or HTML. It offers as well the possibility to replace or alter specific data to correct data entry, configuration or tester interface limitation.

Examinator, optimiSE STDF Explorer and YieldWerx

Galaxy’s Examinator, optimiSE’s STDF Explorer and YieldWerx are all ATE data analysis tools that with more or less features allow visualizing, checking or even editing STDF files.

3.3 Existing solutions and reusing

As seen above, the market offers solutions regarding the handling of STDF files but none of them with the exact same purpose of the problem approached in this project, which, among others, is to convert STDF files into the EXF format. Since such format is property of Infineon Technologies and used internally, it is easy to understand that no solutions for this particular problem exist in the market.

Nevertheless, the EBS system, due to its complexity and the diverse modules, components and functionalities it offers, was programmed with a considerable amount of reusable code that can be extended to new features to be developed. All software development at Critical Software passes through a series of evaluations, part of the Quality Assurance system established in the company. Each of its undergoing projects has its code and documentation cyclically reviewed and approved, not only to assure that what is done is produced with quality according to the client needs but also to improve the system's maintenance and allow effective code reusing.

The EBS Data Loading module is composed by various components – in particular, a set of converters that transform raw data into EXF based information. Due to the significant amount of data types and formats EBS deals with, and since its converters have similar ways of working, there was an urge to standardize the process used to create such converters. A converter template was created, including a set of predefined classes with a basic skeleton to support the standard functionalities of a brand new converter. Besides guaranteeing easier code maintenance, the new converter is, in this way, more likely to behave identically as the other converters, concerning parameters such as functionality and performance.

Although one might think that it could be trivial to implement a new STDF2EXF converter over a predefined structure, the truth is that the existing template was designed having in mind the existing converters in EBS, which handle exclusively text based data, contrary to this new converter that should handle binary data from STDF files. One of the challenges of making a new STDF2EXF converter would be to integrate the existing logic of the STDF Loader – which translates or converts a binary STDF file in memory and loads its data into the system's database – with the converter template, preserving the structure that is used by the existing converters but at the same time keeping the core conversion logic provided by the STDF Loader.

Added to the development of a new converter, one of the most interesting parts of the project would be making the converter work in standalone mode. The existing converters and the STDF Loader are fully integrated with the EBS system, meaning that all their configurations are stored in a repository database – EBS_REPO – and several procedures require a connection to a database in order to obtain information needed to execute their tasks. In standalone mode this is not applicable, since the application would have to work independently, without any database connectivity whatsoever. This would involve incorporating into the converter new mechanisms that would allow it to work differently in both contexts – integrated with EBS DB and in standalone mode – without loosing its purposes in any of the situations.

Besides reusing the STDF Loader's code and making use of the converter template, other existing information could be used in this project, particularly some of the STDF Loader's test cases that were incorporated into the STDF2EXF Converter's test cases catalogue regarding the loading of STDF files into the system by the EXF Loader.

The converter template

The converter template was designed in order to facilitate the development of new converters, being comprised by different files as follows:

- **`_TYPEMakefile`** – Makefile that compiles the converter in the HP-UX operating system.
- **`_TYPE_.cfg`** – Text file with configurations used by the converter to read text raw data files – not relevant to binary converters, since such configurations only apply to text/ASCII structured files.
- **`_TYPE__defines.h`** – Constants used by the converter's classes.
- **`_TYPE_Rawfile.h`** – Structures and vectors common to the class that handles the raw data files conversion process into XML based EXF files.
- **`_TYPE__settings__VERSION__cfg.cfg`** – Text file with configurations used by the converter to read specific versions of text raw files – not relevant to binary converters, since such configurations only apply to text/ASCII structured files.
- **`_TYPE__settings__VERSION_`** – Text file with configurations used by the converter to read specific versions of text raw files – not relevant to binary converters, since such configurations only apply to text/ASCII structured files.
- **`_TYPE__VERSION_Defines.h`** – Constants used by the converter's classes for a specific version of the file type handled by the converter.
- **`_TYPE_Raw_VERSION_File.h`** – Structures and vectors common to the class that handles the conversion process of a specific version's raw data files into XML based EXF files.
- **`_TYPE_Raw_VERSION_File.cpp`** – Handles the conversion process of a specific version's raw data files into XML based EXF files.
- **`_TYPE_2ExfConverterApp.h`** – Header file with functions and variables used by the class that provides methods to support the input and output objects creation.
- **`_TYPE_2ExfConverterApp.cpp`** – Provides methods to support the input and output objects creation.
- **`_TYPE_2EXF.h`** – Header file with methods and functions used by the class that represents the converter's process.
- **`_TYPE_2EXF.cpp`** – Represents the converter's process. A new process is launched every time the converter is called to take action on a file.
- **`_TYPE_2EXF_main.cpp`** – The main class of the converter. It handles the input typed by the user, launches a new converter process and initializes the loggers.

In the file structure above, `_TYPE_` represents the file type – STDF, in this case – whereas `_VERSION_` represents the version of the file format – version 4 would be the current version of STDF. This structure does not need to be strictly followed and it can be adapted according to the different converters' needs.

3.4 Programming languages used in the project

EBS DB is programmed with C++ and PL/SQL (Procedural Language/Structured Query Language). The logic beneath the EBS Data Loading components is coded with C++, whereas the access to the Oracle databases is performed using PL/SQL, Oracle's procedural extension to the SQL database language.

Most of the work developed during the project was achieved using C++ and only a very small portion using PL/SQL. Below is given an overview of these languages, as described respectively in [17] and [18].

C++

C++ is a general-purpose programming language, regarded as a mid-level language, as it comprises a combination of both high-level and low-level language features. It is a statically typed, free-form, multi-paradigm, usually compiled language, supporting procedural programming, data abstraction, object-oriented programming and generic programming.

The language came as an enhancement of the commonly known C language, with the addition of classes, followed by virtual functions, operator overloading, multiple inheritance, templates and exception handling, among other features.

Some of the principles beyond C++ are as follows:

- C++ is designed to directly and comprehensively support multiple programming styles, thus giving the programmer the possibility to choose between its programming or problem approach, even if this makes it possible for the programmer to choose incorrectly;
- C++ is designed to be as compatible with C as possible, therefore providing a smooth transition from C, being as efficient and portable as this language;
- C++ avoids features that are platform specific or not general purpose;
- C++ does not incur overhead for features that are not used;
- C++ is designed to function without a sophisticated programming environment.

C++ standard consists of the core language and the C++ standard library. The latter includes most of the Standard Template Library (STL) and a slightly modified version of the C standard library. The language provides useful tools as containers (such as vectors or lists), iterators to provide these containers with array-like access, and algorithms to perform operations such as searching and sorting.

PL/SQL

PL/SQL is one of three languages embedded in the Oracle Database, the other two being SQL and Java. It supports variables, conditions, arrays and exceptions. Implementations from version 8 of Oracle Database onwards have included features associated with object-orientation.

The language allows defining classes and instantiating these as objects in PL/SQL code, resembling object-oriented programming languages like Object Pascal, C++ and Java.

PL/SQL functions analogously to the embedded procedural languages associated with other relational databases, such as Sybase ASE and Microsoft SQL Server which use Transact-SQL, or PostgreSQL which uses PL/pgSQL (tries to emulate PL/SQL to an extent).

4 Specification of the new solution

After gaining business know-how on the semiconductor manufacturing process and on the EBS system, two documents were written as the base of the work to be done throughout the project.

A Software Requirements Specification [11] and a Test Case Specification [19] were written based on templates previously produced in Critical Software's Quality department. From the needs of the application a requirements catalogue was gathered prior to the development phases, comprising functional and non-functional software requirements with the purpose of guiding in a correct way the developer when engineering the application. The former document includes as well a use cases catalogue – comprising an actors list and a use cases list –, a domain model, together with a state and sequence diagrams. Test cases were written in the latter document and updated throughout the project, to assure that what would be implemented could be tested in accordance to the specified requirements.

The use of such documents obliged one to understand the problem's nature, by establishing, in a methodical way, the different needs of the application and its structure, guiding the trainee to start developing the converter with the conscience of what needed to be done and how.

The requirements and diagrams shown in the following pages were designed using Enterprise Architect and the UML (Unified Modeling Language) language.

4.1 Software requirements catalogue

This catalogue is divided by requirements type, where functional requirements are presented first, followed by non-functional requirements – implementation, interoperability and validation. There was not a need to establish further requirements, as the new converter gathered most of its functionality from the STDF Loader component, an existing application with a known purpose and defined requirements.

Functional requirements

Below are described the functional requirements for the STDF2EXF Converter as defined in the Software Requirements Specification [11].

STDF2EXF-SRS-FUN-001 Data conversion							
Type	«Functional»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 12-10-2007. Modified on 12-10-2007.						
The STDF2EXF Converter shall transform the binary data of a valid input STDF file into text data stored in a new output XML file.							

STDF2EXF-SRS-FUN-002 Working modes	
Type	«Functional»
Status	Proposed

Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 12-10-2007. Modified on 12-12-2007.						

The STDF2EXF Converter shall have two working modes:

- integrated in EBS, as a standard converter, communicating with the other system components and loading information from one or more databases;
- in standalone mode, independently from an EBS installation, not accessing any database and being configured by reading .ini files.

In order to work in standalone mode, the STDF2EXF Converter should be called in the command line, with an input STDF work package path as argument, as follows:

- STDF2EXF -s {work package path}.

STDF files contain binary data, which is difficult to interpret and maintain; by running the converter in standalone mode it is possible to quickly access the information of a STDF file in text format by opening the generated XML file, apart from the EBS workflow process.

STDF2EXF-SRS-FUN-003 Configuration							
Type	«Functional»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 12-10-2007. Modified on 12-12-2007.						

The converter shall have one or more .ini files in '/bin/config', containing configurations for its Basic Setup, Logger and WP Logger. This information shall only be accessed when running in standalone mode, otherwise the converter shall get its configurations from the EBS_REPO database.

STDF2EXF-SRS-FUN-004 Logging							
Type	«Functional»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 19-10-2007. Modified on 12-12-2007.						

For each day the STDF2EXF Converter is used, a log containing its most relevant activity should be created.

The logfile should be named stdf2exf_{timestamp}.log and be stored in the folder '/bin/log'.

The {timestamp} value shall have the format YYYYMMDD,
with YYYY being the year, MM the month and DD the day.

Each work package shall have its own log as well.

STDF2EXF-SRS-FUN-006 Converter help							
Type	«Functional»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 22-10-2007. Modified on 22-10-2007.						

The converter shall display information regarding its version and working modes. The user shall access this information when calling the application using the command STDF2EXF {argument}. The parameter {argument} can have the following values: - "-V" for version; - "--help" for help.

Implementation requirements

The following requirements can be seen as non-testable and non-functional requirements, regarding the implementation of the STDF2EXF Converter.

STDF2EXF-SRS-IMP-001 STDF2EXF Converter structure							
Type	«Implementation»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 12-10-2007. Modified on 12-10-2007.						
The STDF2EXF Converter shall follow the same template used to build the other converters of the EBS system. Therefore, this converter shall have the same structure as the other ones, being its logic adapted to handle STDF files. Although each converter handles differently its data type, all follow the same pattern, being built from a single template, which allows converters to be built and work in a similar way.							

STDF2EXF-SRS-IMP-002 File naming convention							
Type	«Implementation»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 19-10-2007. Modified on 19-10-2007.						
The generated XML files shall have the name: EXF_{input_file_name}.{old_extension}.xml. Example: Input STDF file: STDF_test.std -> Output XML file: EXF_STDF_test.std.xml							

Interoperability requirements

The requirement shown below describes how the STDF2EXF Converter shall interact with other applications or components of EBS Data Loading.

STDF2EXF-SRS-INT-001 Integration with EBS							
Type	«Interoperability»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 12-10-2007. Modified on 12-12-2007.						
If not in standalone mode, the STDF2EXF Converter shall work integrated in EBS, communicating through Yoda messages with the JM Emulator. As other converters, the STDF2EXF Converter should communicate with the JM Emulator by Yoda messages, sending "wait" messages and signalling the status of its tasks.							

Validation requirements

Below are presented the non-functional requirements concerning the validity of the files handled and generated by the STDF2EXF Converter.

STDF2EXF-SRS-VAL-001 STDF validation							
Type	«Validation»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 12-10-2007. Modified on 12-12-2007.						
The STDF2EXF Converter shall only process valid STDF files from its input folder.							

STDF2EXF-SRS-VAL-002 XML structure validation							
Type	«Validation»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 12-10-2007. Modified on 12-12-2007.						
The output XML file shall be built from and validated against a predefined EXF schema.							

STDF2EXF-SRS-VAL-003 XML data integrity							
Type	«Validation»						
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**	Medium	**Difficulty**	Medium
Effort							
Details	Created on 16-10-2007. Modified on 16-10-2007.						
The content of the generated output XML file shall not be corrupted, regarding the original data of the input STDF file.							

4.2 Use cases catalogue

The next pages describe in detail the different use cases and the actors that interact with the application as defined in the Software Requirements Specification [11].

Figure 13 shows the UML use case diagram for the STDF2EXF Converter followed by its description.

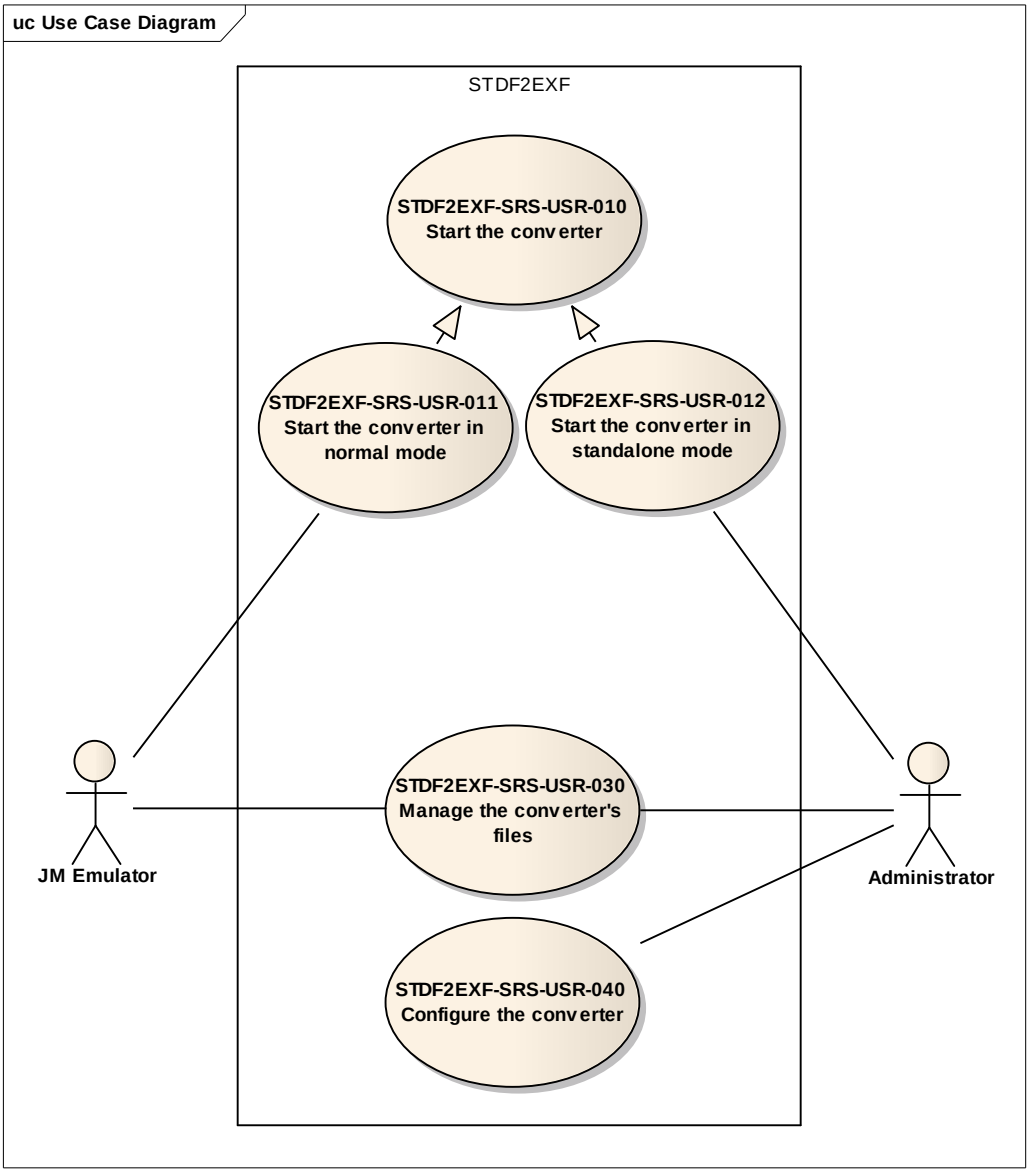

Figure 13: Use case diagram

Actors list

Actors are the individuals or entities that interact with the system. They have an important role as they are directly related to how a system behaves and the system itself shall be built so that the interaction between actors and system is optimal. Two actors can be found in EBS DB, within the context of the STDF2EXF Converter project: the Administrator and the JM Emulator. Both actors are described in the following paragraphs.

Administrator

The administrator uses and manages EBS DB. Besides launching the JM Emulator, thus initializing the EBS Data Loading workflow, this actor might be interested in using the STDF2EXF Converter in standalone mode, with the goal of quickly converting and accessing the information of a STDF file. The administrator shall be responsible for changing the converter's standalone configuration settings.

JM Emulator

The JM Emulator controls the EBS Data Loading process, communicating with the different EBS DB components, namely with the STDF2EXF Converter, by sending and receiving YODA messages. It is also responsible for starting the STDF2EXF Converter, moving the work package files between the converter's folders and executing a controlled shutdown of the component.

Use cases

Use cases describe the interaction between actor and system, represented as a sequence of simple steps in order to describe possible scenarios of interactivity in accordance to the defined requirements.

As shown in Figure 13, five use cases were considered for the STDF2EXF Converter. Each of these is described in the following tables.

STDF2EXF-SRS-USR-010 Start the converter							
Type							
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**		**Difficulty**	
Effort							
Details	Created on 11-10-2007. Modified on 11-10-2007.						
The converter is started by the JM Emulator component, or is manually started by the administrator, in standalone mode.							

STDF2EXF-SRS-USR-011 Start the converter integrated with EBS							
Type							
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**		**Difficulty**	
Effort							
Details	Created on 11-10-2007. Modified on 11-10-2007.						
The converter is started as part of the EBS workflow by the JM Emulator.							

STDF2EXF-SRS-USR-012 Start the converter in standalone mode							
Type							
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**		**Difficulty**	
Effort							
Details	Created on 11-10-2007. Modified on 11-10-2007.						
An administrator manually starts the STDF2EXF Converter in standalone mode.							

STDF2EXF-SRS-USR-030 Manage the converter's files							
Type							
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**		**Difficulty**	
Effort							
Details	Created on 11-10-2007. Modified on 20-12-2007.						
The JM Emulator adds, removes or moves files between the converter's folders, before and after STDF processing. The administrator manually adds, removes or moves files from the converter's folders, to process a STDF work package of his choice.							

STDF2EXF-SRS-USR-040 Configure the converter							
Type							
Status	Proposed						
Version	1.0	**Phase**	1.0	**Priority**		**Difficulty**	
Effort							
Details	Created on 11-10-2007. Modified on 20-12-2007.						
The administrator configures the converter by changing its basic setup and its loggers.							

4.3 Domain model

The domain model for the STDF2EXF Converter, as defined in the Software Requirements Specification [11], has the goal of giving an overall idea of how the converter classes shall be organised and relate to each other. The UML class diagram is displayed in Figure 14.

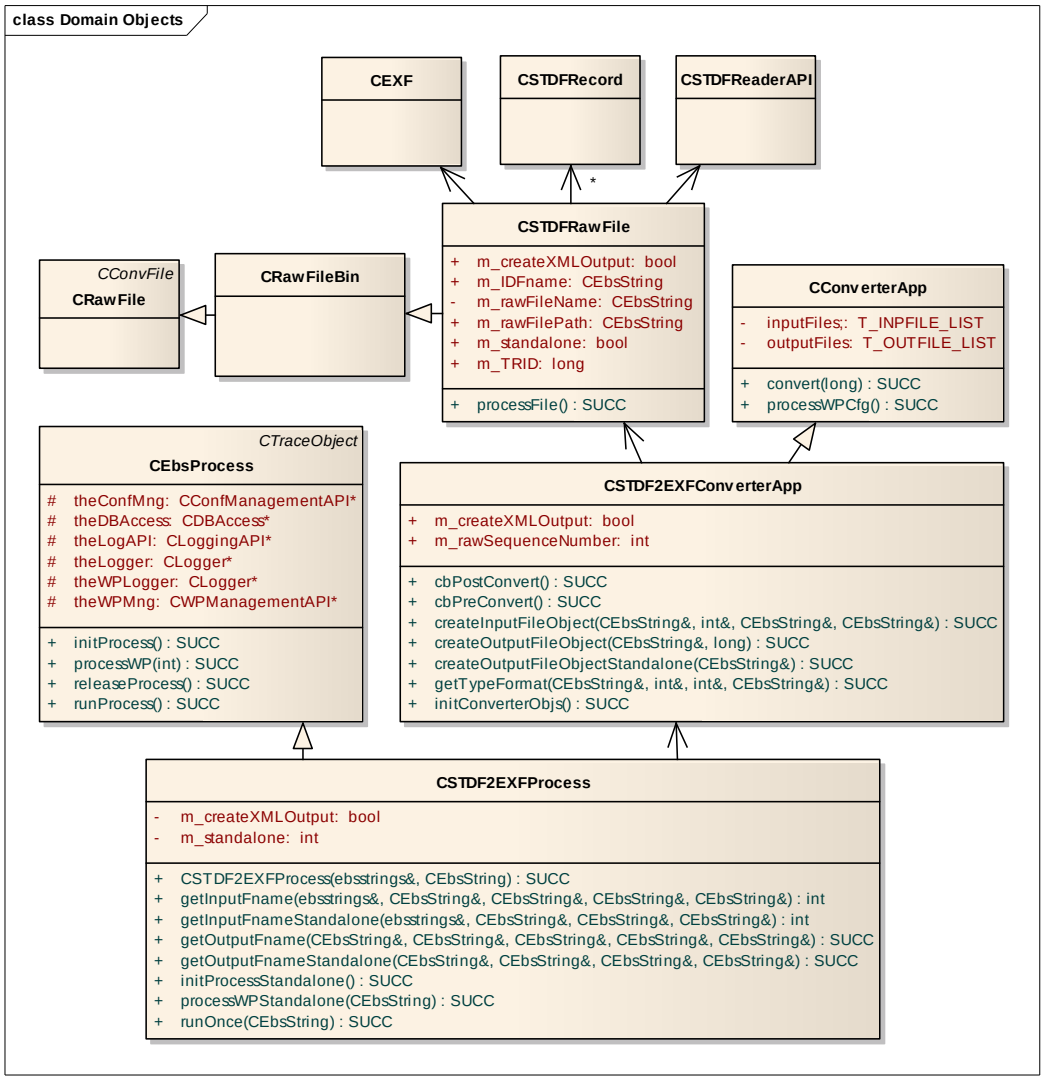

Figure 14: Class diagram for the STDF2EXF Converter

This diagram shows in some detail the three main classes of the converter (`CSTDF2EXFProcess`, `CSTDF2EXFConverterApp`, `CSTDFRawFile`), plus the ones that are directly related to these, as written in the next pages.

CSTDF2EXFProcess

`CSTDF2EXFProcess` should be the main class of the converter, being responsible for its main processing. It should handle the creation of the work package related records, input and output file objects. To achieve this, it should use the converter framework, which is a set of classes that supports raw and configuration file reading, together with EXF file writing functionalities.

When integrated with EBS, this class should handle the converting process, regarding the following steps:

1. Retrieve the settings from the repository database – EBS_REPO – for the converter and IDF API (Application Programming Interface)[3];
2. Find a WP to convert and perform the following steps for its raw file:
    a. Create a new JM_WPS record in the repository database with converter and WP information;
    b. Create new input and output file objects to handle the input and output files;
    c. Start the conversion process.
3. If ready, store the WP properties in the repository database and find the next WP to convert.

In standalone mode, the following should be done:

1. Retrieve the settings from local configuration files for the converter and IDF API;
2. Perform the following steps for the WP passed as argument by the user:
    a. Create new input and output file objects to handle the input and output files;
    b. Start the conversion process.

CSTDF2EXFConverterApp

`CSTDF2EXFConverterApp` should provide methods to support the input and output object creating. It should be a child class of the converter framework.

CSTDFRawFile

`CSTDFRawFile` should convert the STDF files into EXF files and implement the business logic of the converter.

CEBSProcess

`CEBSProcess` represents a process in EBS. Each EBS component inherits methods from this class.

[3] The IDF API contains methods that support the handling of IDF files. Since EXF is an Intermediate Data Format, the IDF API is used by the EBS components to access the information contained in such files.

CTraceObject

`CTraceObject` represents a root object for class-level tracing.

CConverterApp

`CConverterApp` is the super class used by all the converters of EBS DB.

CRawFileBin

`CRawFileBin` handles the opening of a binary file. It is used by the STDF Loader and shall be incorporated into the STDF2EXF Converter.

CRawFile

`CRawFile` is the super class of the `CRawFileBin` and `CRawFileAscii` (used similarly to `CRawFileBin` for text files) classes.

CConvFile

`CConvFile` is the super class of the `CRawFile` class.

Together with the UML class diagram for the STDF2EXF Converter, another diagram was produced regarding the methods that should be added to the `CLoaderStg` class of the EXF Loader component. Figure 15 shows the main classes of the EXF Loader and, more importantly, the methods that should be added or changed in this component so that it can support STDF files.

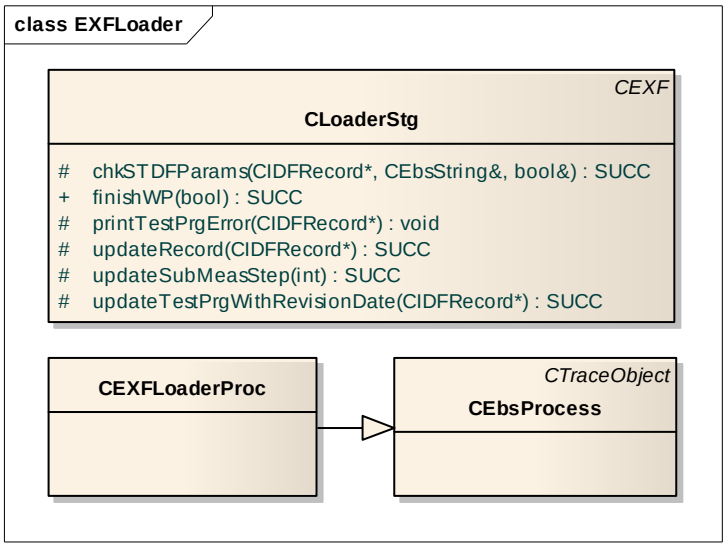

Figure 15: Class diagram for the EXF Loader (changes to be made only)

The EXF Loader's `CLoaderStg` class is responsible for processing the EXF files that passed through the different converters and have been transferred to the EXF Loader's input folder, so that their data can be loaded into the staging database. Since the STDF Loader overrides this process for STDF files, there would be a need to incorporate some of the STDF handling

logic of the STDF Loader into the EXF Loader. This should be done by updating the functions shown in the picture above, if they exist, or migrate them to this class.

4.4 Sequence diagrams

The project's UML sequence diagrams catalogue has the goal of showing the STDF2EXF Converter's sequence of main actions in its different usage scenarios.

Figure 16 shows the sequence diagram for the STDF2EXF Converter when running integrated with EBS DB.

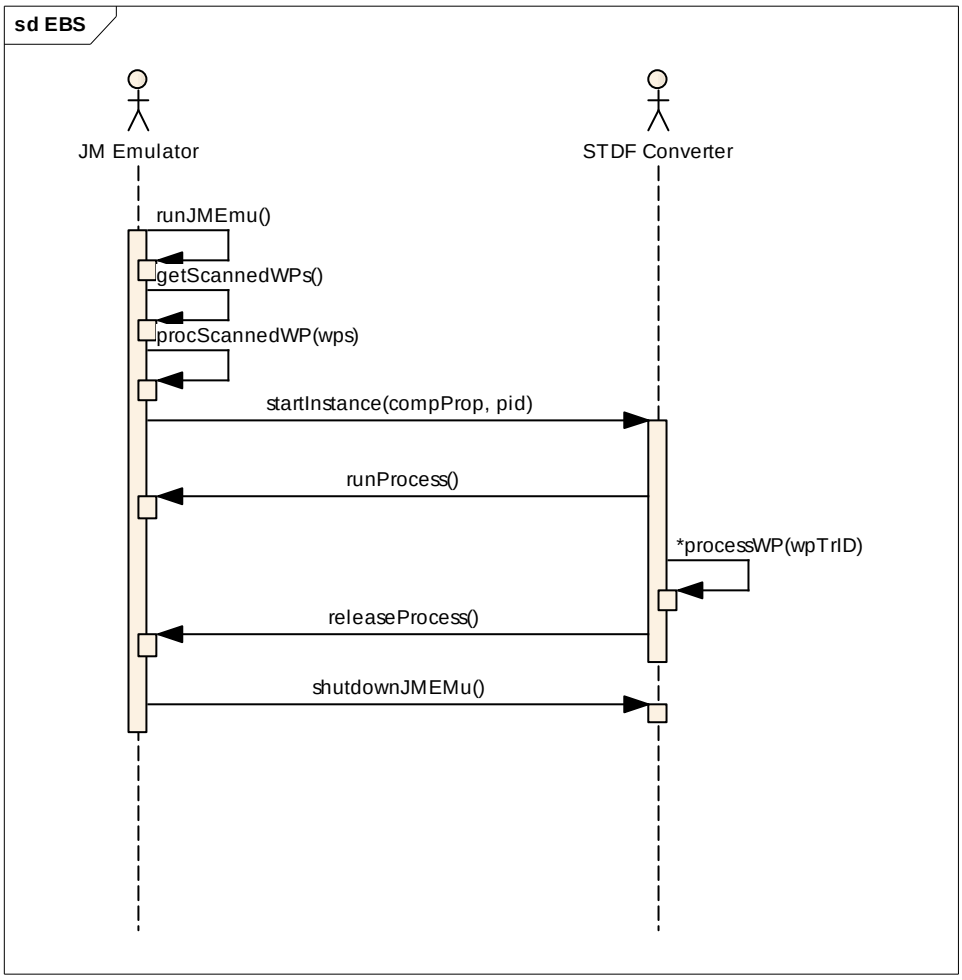

Figure 16: The STDF2EXF Converter when running integrated with EBS DB

As seen above, the JM Emulator is started by an administrator, launching its own process in a never ending loop. The JM Emulator retrieves the work packages that have been scanned from the WP Scanner's input folders and places them in the input folder of the appropriate converter. If there are STDF files waiting to be converted, the JM Emulator starts an instance of the STDF2EXF Converter, so that this component processes the files placed in its input folder, converting them into EXF files, ready to be loaded by the next component in the workflow – EXF Loader – into the EBS_STAGE database. The STDF2EXF Converter waits until there are no more work packages to process and shuts itself down after a timeout period. In a similar way, the JM Emulator shuts itself down after a period of inactivity.

Figure 17 shows the sequence diagram for the STDF2EXF Converter when running independently from EBS, in standalone mode.

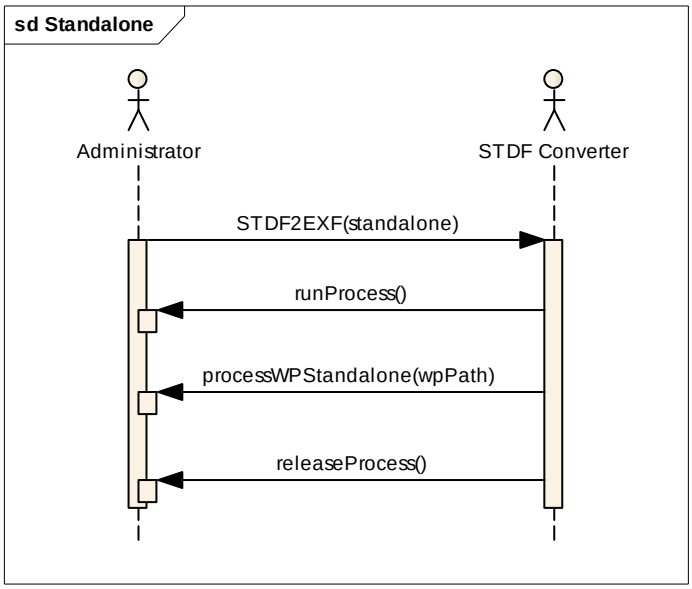

Figure 17: The STDF2EXF Converter when running in standalone mode

In this case, the administrator calls the converter with two arguments: one indicating it will run in standalone mode and the other one with the work package to process. The application converts the STDF raw file contained in the work package and generates a EXF file that should be placed in the same folder. The converter terminates when conversion is finished and frees its allocated resources.

4.5 State diagrams

UML state diagrams have the goal of showing the system's behaviour through the presentation of its states when specific sequences of actions are carried out.

To allow one to understand the state diagram presented in this section, it might be useful to have an insight of how EBS work packages are prioritized and how this is achieved using process streams. Work packages are processed using streams that have been defined in the repository database; one stream can be associated with one or more file types, meaning that one stream can process STDF, TDF and APRC files at the same time, for example, while one file type can as well be associated with several streams, thus being possible to process the same number of work packages of a same file type in parallel as the number of streams that have such file type associated. Figure 18 shows an example of how the streams can be defined in EBS DB.

	STM_ID	STM_PARTITION_ID	STM_TYPE	STM_DATA_TYPE	STM_STATE	STM_TR_ID
1	1	1	STDF Stream	TRIGGER, STDFLoader	ACTIVE	(null)
2	2	2	STDF Stream	TRIGGER, STDFLoader	ACTIVE	(null)
3	3	3	STDF Stream	TRIGGER, STDFLoader	ACTIVE	(null)
4	4	4	STDF Stream	TRIGGER, STDFLoader	ACTIVE	(null)
5	5	5	STDF Stream	CP, STDFLoader	ACTIVE	(null)
6	6	6	STDF Stream	CP, STDFLoader	ACTIVE	(null)
7	7	7	Etc	2DPATREC, APRC, FAB300EDC, ChipXRef, LDL, TDF, PCMDD_LIMIT, PCMDD_OUT, YIELDRU...	ACTIVE	(null)
8	8	8	STDF Stream	STDFLoader	INACTIVE	(null)

Figure 18: JM streams

In the figure above it is possible to see that STDF files can be processed in parallel for a maximum of six files, corresponding to the six streams allocated to the STDF Loader. On the contrary, the processing of the file types associated to stream 7 has less priority, since a single stream has been defined to handle several file types, meaning that files of such type are processed sequentially and never in parallel in this case.

Figure 19 presents the state diagram for the STDF2EXF Converter, when running integrated with the EBS system (no state diagram for the standalone mode is presented, as the behaviour of the application in this case is trivial and none of the concepts mentioned in this subchapter apply to it).

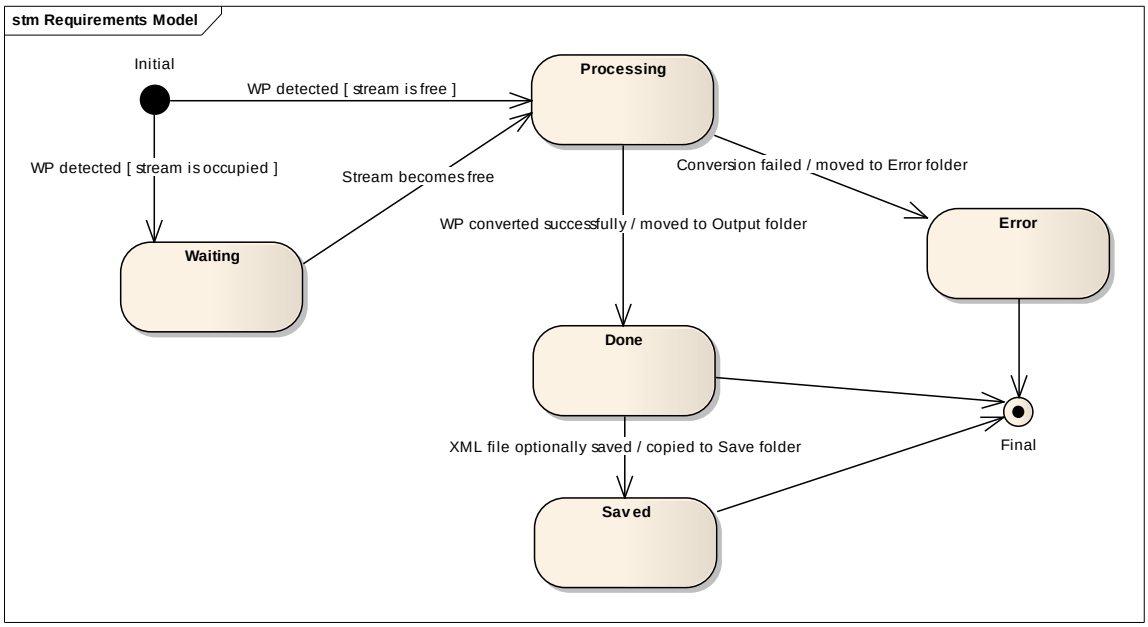

Figure 19: State diagram for the STDF2EXF Converter (not applicable to standalone mode)

When a work package is detected in the STDF2EXF Converter's input folder, an action will be carried on in order to handle the work package. The work package will be immediately processed if an associated stream is free (not processing any work package at that time), or waits until a stream which is currently busy finishes its task. Once the processing is finished, the work package is moved to the converter's output folder or, if an error occurs, moved into an error folder. The generated EXF file might also be optionally saved into a proper folder (*/save*) in case a flag has been activated for that purpose.

4.6 Test cases catalogue

Test cases allow one to certify that the application is designed according to the requirements that were specified in the beginning and throughout the project.

The converter should be tested according to the following items:

- Working modes;
- Configuration;
- Conversion;
- Validation;
- Logging;
- Structure;
- Integration with the EXF Loader;
- Loading parameter checking;
- Rosetta Net support.

Once the STDF2EXF Converter would be fully integrated with the EXF Loader, the latter should be tested based on the STDF Loader's loading test cases, which were modified and included in the STDF2EXF Converter's test cases catalogue [19].

All non-measurable non-functional requirements did not require testing, i.e., implementation, interoperability and validation requirements.

Below are described the test cases for the STDF2EXF Converter, divided by modules. The relation between these test cases and requirements or use cases can be seen in Chapter 4.7, Traceability matrix.

Working modes

Standalone mode	
Test case identifier:	STDF2EXF-TCS-WRK-001 Standalone mode
Responsibility:	Critical Software
Purpose:	This test case shall verify if the STDF2EXF Converter runs in standalone mode.
Author(s):	Daniel Silva
Input Specifications	
a. Run the STDF2EXF Converter, by calling it using the command line as follows: > STDF2EXF –s {work package path}	
Output Specifications	
a. The STDF2EXF Converter shall recognize the typed command and execute in standalone mode.	
Other N/A	
Dependencies N/A	

Configuration

INI file recognition (success case)	
Test case identifier:	STDF2EXF-TCS-CFG-001 INI file recognition (success case)
Responsibility:	Critical Software
Purpose:	This test case shall verify if the STDF2EXF Converter recognizes a valid configuration file.
Author(s):	Daniel Silva
Input Specifications	
a. Make sure the files "stdf2exf_basic_setup.ini", "stdf2exf_logger.ini" and "stdf2exf_wplogger.ini" are in the converter's "bin/config" folder and they have the correct structure and parameters. b. Run the STDF2EXF Converter in standalone mode.	
Output Specifications	
a. The converter shall not generate any error message regarding its configuration and shall proceed with its execution.	
Other N/A	
Dependencies STDF2EXF-TCS-WRK-001 Standalone mode, STDF2EXF-TCS-VAL-001 STDF recognition (success case), STDF2EXF-TCS-VAL-005 Raw files validation (success case), STDF2EXF-TCS-VAL-007 Work package validation (success case)	

INI file recognition (fail case)	
Test case identifier:	STDF2EXF-TCS-CFG-002 INI file recognition (fail case)
Responsibility:	Critical Software
Purpose:	This test case shall verify if the STDF2EXF Converter recognizes an invalid configuration file.
Author(s):	Daniel Silva
Input Specifications	
a. Move, rename or introduce errors in the files "stdf2exf_basic_setup.ini", "stdf2exf_logger.ini" or "stdf2exf_wplogger.ini" in the converter's "bin/config" folder. b. Run the STDF2EXF Converter in standalone mode.	
Output Specifications	
a. The converter shall generate an error message regarding its configuration and terminate.	
Other N/A	
Dependencies STDF2EXF-TCS-WRK-001 Standalone mode, STDF2EXF-TCS-VAL-001 STDF recognition (success case), STDF2EXF-TCS-VAL-005 Raw files validation (success case), STDF2EXF-TCS-VAL-007 Work package validation (success case)	

Conversion

Data conversion	
Test case identifier:	STDF2EXF-TCS-CON-001 Data conversion
Responsibility:	Critical Software
Purpose:	This test case shall verify if the STDF2EXF Converter generates a XML file from a valid input STDF file.
Author(s):	Daniel Silva
Input Specifications	

a. Introduce a valid STDF work package (exactly one STDF raw file) into the folder "$EBS_HOME/ data/wpscan/input/STDF". b. Run the JM Emulator. c. Run the STDF2EXF Converter in standalone mode with the work package described in a) as argument.
Output Specifications
a. For step b),the application shall save a XML file in the "$EBS_HOME/ data/converters/STDF2EXF/output" folder with the same name of the input raw file, preceded by "EXF_". b. For step c), the application shall save a XML file in the work package's folder with the same name of the input raw file, preceded by "EXF_".
Other N/A
Dependencies STDF2EXF-TCS-WRK-001 Standalone mode, STDF2EXF-TCS-CFG-001 INI file recognition (success case), STDF2EXF-TCS-VAL-001 STDF recognition (success case), STDF2EXF-TCS-VAL-005 Raw files validation (success case), STDF2EXF-TCS-VAL-007 Work package validation (success case)

Validation

STDF recognition (success case)	
Test case identifier:	STDF2EXF-TCS-VAL-001 STDF recognition (success case)
Responsibility:	Critical Software
Purpose:	This test case shall verify if the converter recognizes valid STDF raw files.
Author(s):	Daniel Silva
Input Specifications	
a. Introduce a work package with a valid STDF file in the converter's input folder ("$EBS_HOME/ data/wpscan/input/STDF"). b. Run the JM Emulator. c. Run the STDF2EXF Converter in standalone mode with the work package described in a) as argument.	
Output Specifications	
a. The Converter shall not generate any error message regarding the recognition of the input STDF file, after b) and c),.	
Other N/A	
Dependencies STDF2EXF-TCS-WRK-001 Standalone mode, STDF2EXF-TCS-VAL-007 Work package validation (success case), STDF2EXF-TCS-VAL-005 Raw files validation (success case)	

STDF recognition (fail case)	
Test case identifier:	STDF2EXF-TCS-VAL-002 STDF recognition (fail case)
Responsibility:	Critical Software
Purpose:	This test case shall verify if the converter recognizes invalid STDF raw files.
Author(s):	Daniel Silva
Input Specifications	
a. Introduce a work package with an invalid or non STDF file in the converter's input folder ("$EBS_HOME/ data/wpscan/input/STDF"). b. Run the JM Emulator. c. Run the STDF2EXF Converter in standalone mode with the work package described in a) as argument.	
Output Specifications	
a. The Converter shall fail, after b) and c), to recognize the input raw file, generating an error message and terminating.	
Other N/A	
Dependencies STDF2EXF-TCS-WRK-001 Standalone mode, STDF2EXF-TCS-VAL-007 Work package validation (success case), STDF2EXF-TCS-VAL-005 Raw files validation (success case)	

XML validation	
Test case identifier:	STDF2EXF-TCS-VAL-003 XML structure validation
Responsibility:	Critical Software
Purpose:	This test case shall verify if the structure of the XML files generated by the STDF2EXF Converter is valid.
Author(s):	Daniel Silva
Input Specifications	
a. Introduce a work package with a valid STDF file in the converter's input folder ("$EBS_HOME/ data/wpscan/input/STDF"). b. Run the JM Emulator. c. Run the STDF2EXF Converter in standalone mode with the work package described in a) as argument. d. Validate the XML files generated in b) and c), against the predefined XML schema ("EXF_Schema_qualified.xsd"), using an XML validator.	
Output Specifications	
a. No errors shall occur during the XML validation.	
Other N/A	
Dependencies STDF2EXF-TCS-CON-001 Data conversion	

XML data integrity	
Test case identifier:	STDF2EXF-TCS-VAL-004 XML data integrity
Responsibility:	Critical Software
Purpose:	This test case shall verify if the data of the generated XML file was not corrupted during the conversion process.
Author(s):	Daniel Silva
Input Specifications	
a. Activate the STDF_CREATE_XML flag in the STDF Loader (using the current system). b. Introduce the regression tests' STDF work packages into the input folder of the STDF Loader ("$EBS_HOME/ data/loader/STDF/input"). c. Start the JM Emulator step by step until the WP has finished its processing in the STDF Loader. d. Save the created EXF file (located in "$EBS_HOME/ data/loader/STDF/output"). e. Introduce the regression tests' STDF work packages into the STDF2EXF's input folder ("$EBS_HOME/ data/wpscan/input/STDF"). f. Run the STDF2EXF Converter in standalone mode with the work package described in e) as argument, and save the created EXF file. g. Compare the generated XML files from steps d) and f).	
Output Specifications	
a. The data of both XML files shall be similar (with different timestamps). Differences should be justified.	
Other N/A	
Dependencies STDF2EXF-TCS-CON-001 Data conversion	

Raw files validation (success case)	
Test case identifier:	STDF2EXF-TCS-VAL-005 Raw files validation (success case)
Responsibility:	Critical Software
Purpose:	This test case shall verify if the converter only processes work packages that have exactly one raw file.
Author(s):	Daniel Silva
Input Specifications	
a. Introduce a work package with exactly one raw file into the converter's input folder ("$EBS_HOME/ data/wpscan/input/STDF"). b. Run the JM Emulator.	

c. Run the STDF2EXF Converter in standalone mode with the work package described in a) as argument.
Output Specifications
a. The converter shall recognize, after b) and c), that the work package has one raw file and execute normally.
Other
N/A
Dependencies
STDF2EXF-TCS-WRK-001 Standalone mode, STDF2EXF-TCS-VAL-007 Work package validation (success case)

Raw files validation (fail case)	
Test case identifier:	STDF2EXF-TCS-VAL-006 Raw files validation (fail case)
Responsibility:	Critical Software
Purpose:	This test case shall verify if the converter does not process work packages that have zero or more than one raw files.
Author(s):	Daniel Silva
Input Specifications	
a. Introduce a work package with zero or more than one raw files into the converter's input folder ("$EBS_HOME/ data/wpscan/input/STDF"). b. Run the JM Emulator. c. Run the STDF2EXF Converter in standalone mode with the work package described in a) as argument.	
Output Specifications	
a. The converter shall recognize, after b) and c), that the work package has zero or more than one raw files and exit with an error message.	
Other	
N/A	
Dependencies	
STDF2EXF-TCS-WRK-001 Standalone mode, STDF2EXF-TCS-VAL-007 Work package validation (success case)	

Work package validation (success case)	
Test case identifier:	STDF2EXF-TCS-VAL-007 Work package validation (success case)
Responsibility:	Critical Software
Purpose:	This test case shall verify if the converter recognizes a valid work package path.
Author(s):	Daniel Silva
Input Specifications	
a. Start the converter in standalone mode: > STDF2EXF –s {valid work package path}	
Output Specifications	
a. The converter shall detect a valid work package path and execute normally.	
Other	
N/A	
Dependencies	
STDF2EXF-TCS-WRK-001 Standalone mode	

Work package validation (fail case)	
Test case identifier:	STDF2EXF-TCS-VAL-008 Work package validation (fail case)
Responsibility:	Critical Software
Purpose:	This test case shall verify if the converter recognizes an invalid work package path.
Author(s):	Daniel Silva
Input Specifications	
a. Start the converter in standalone mode: > STDF2EXF –s {invalid work package path}	
Output Specifications	

a. The converter shall detect an invalid work package path and exit with an error message.
Other N/A
Dependencies STDF2EXF-TCS-WRK-001 Standalone mode

Logging

Logging	
Test case identifier:	STDF2EXF-TCS-LOG-001 Logging
Responsibility:	Critical Software
Purpose:	This test case shall verify if a log file is created or updated after an execution of the STDF2EXF Converter.
Author(s):	Daniel Silva
Input Specifications	
a. Run the JM Emulator. b. Run the STDF2EXF Converter in standalone mode.	
Output Specifications	
a. A non-empty log file named stdf2exf_<yyyy><mm><dd>.log should exist in the application's /log folder. Another log file shall exist in the folder of the processed work package.	
Other N/A	
Dependencies STDF2EXF-TCS-WRK-001 Standalone mode	

Structure

Structure	
Test case identifier:	STDF2EXF-TCS-STR-001 Structure
Responsibility:	Critical Software
Purpose:	This test case shall verify if the converter's structure follows the one defined in the templates designed for the construction of new converters.
Author(s):	Daniel Silva
Input Specifications	
a. Open the converter's source folder.	
Output Specifications	
a. The STDF2EXF Converter shall contain, at least, the following files (or similar): • STDFdefines.h • STDF2EXF.h • STDF2ExfConverterApp.h • STDFRawFile.h • STDF2EXF_main.cpp • STDF2EXF.cpp • STDF2ExfConverterApp.cpp • STDFRawFile.cpp • Makefile	
Other N/A	
Dependencies N/A	

Integration with the EXF Loader

The following test cases were not developed during the project's period and due to this fact are not included in 4.7 Traceability matrix. Nevertheless, such test cases are necessary to test the converter's integration with the EXF Loader.

STDF. LOAD.1	
Test case identifier:	STDF. LOAD.1
Responsibility:	Critical Software
Purpose:	*Loading STDF files into the DB*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully • Oracle schemas shall be empty	
Input Specifications	
• STDF_TEST_WP_01, STDF_TEST_WP_04, STDF_TEST_WP_07, STDF_TEST_WP_09, STDF_TEST_WP_10, STDF_TEST_WP_11, STDF_TEST_WP_12, STDF_TEST_WP_13, STDF_TEST_WP_15, STDF_TEST_WP_16, STDF_TEST_WP_17, STDF_TEST_WP_18, STDF_TEST_WP_19, STDF_TEST_WP_20, STDF_TEST_WP_21, STDF_TEST_WP_22, STDF_TEST_WP_23, STDF_TEST_WP_24, STDF_TEST_WP_25, STDF_TEST_WP_26, STDF_TEST_WP_27, STDF_TEST_WP_28, STDF_TEST_WP_29, STDF_TEST_WP_31, STDF_TEST_WP_32, STDF_TEST_WP_33, STDF_TEST_WP_34, STDF_TEST_WP_36, STDF_TEST_WP_37, STDF_TEST_WP_38, STDF_TEST_WP_39, STDF_TEST_WP_40, STDF_TEST_WP_41, STDF_TEST_WP_42, STDF_TEST_WP_43, STDF_TEST_WP_44, STDF_TEST_WP_45, STDF_TEST_WP_48, STDF_TEST_WP_49, STDF_TEST_WP_50, STDF_TEST_WP_51, STDF_TEST_WP_52, STDF_TEST_WP_54, STDF_TEST_WP_55, STDF_TEST_WP_58, STDF_TEST_WP_59 STDF work packages	
Actions Specification	
• Start the JM-Emu if it is not running • Copy the STDF work packages into the STDF2EXF Converter's input	
Output Specifications	
• The STDF2EXF Converter shall load all work packages successfully	
Post-Conditions:	
None	
Dependencies	

STDF. LOAD.2	
Test case identifier:	STDF. LOAD.2
Responsibility:	Critical Software
Purpose:	*The loader checks the existence of the appropriate test program entry. If it does not exist, it is an error*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully • Oracle schemas shall be empty • The configs table flag InsertNewTestprogram shall be Off	
Input Specifications	
• Content of the STDF_TEST_WP_01 directory from the test archive	
Actions Specification	
• Set the InsertNewTestprogram flag to off using the following script:	

```
cd $TEST_CASES_DIR/STDF2EXF.inp/sql_scripts
sqlplus
$REPOSITORY_SCHEMA_NAME/$REPOSITORY_SCHEMA_PASSWORD@CONN_STR_REP
O
```

```
@insertNewTestProgramOff.sql
```

- Copy the STDF_TEST_WP_01 into the STDF2EXF Converter input component.

```
cd $TEST_CASES_DIR
cp –r STDF_TEST_WP_01 $EBS_HOME/data/wpscan/input/STDF
jm –askme
```

Output Specifications

- The status of the WP shall be ERROR.
- In the log file an error message should be present stating that no Testprograms record was present in the database.

Post-Conditions:

None

Dependencies

STDF. ITMFG5312	
Test case identifier:	STDF. ITMFG5312
Responsibility:	Critical Software
Purpose:	*Artificial Wafer numbers for Tracking Unit record*
Author(s):	Norberto Leite, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- Oracle schemas shall be empty

Input Specifications

- TDF_EZ18A90L_S11P TDF WP
- Content of the STDF_TEST_WP_L01 directory from the test archive

Actions Specification

- Start the JM-Emu if it is not running
- Copy the TDF masterdata into the proper WPScan's input
- Wait until the masterdata are loaded
- Insert a new record into the Tracking_Units table:

```
sqlplus <ebs_stage>/<ebs_stage>@<conn_string>
SQL> @ STDF.ITMFG5312.sql
```

- Copy the STDF WP into the STDF2EXF Converter's input
- Wait until the STDF WP is loaded
- Copy the STDF WP again into the STDF2EXF Converter's input
- Wait until the STDF WP is loaded again

Output Specifications

- The <EBS_ANA>.TRACKING_UNITS table shall contain one record:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT id, motherlot_id, wafernumber, wafer_id FROM
tracking_units;

ID MOTHERLOT_ID  WAFERNUMBER WAFER_ID
-- ------------- ----------- ----------
 1      EZ18A90L           2   K102GKN0
```

- The <EBS_ANA>.MEASUREMENTS table shall contain two records:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT id, tunit_id, meas_counter FROM measurements WHERE
component_seq = 0 ORDER BY meas_counter;

 ID   TUNIT_ID MEAS_COUNTER
--- ---------- ------------
  1          1            1
```

2	1	2

Post-Conditions:

None

Dependencies

STDF. CQ1037	
Test case identifier:	STDF. CQ1037
Responsibility:	Critical Software
Purpose:	*LimitFileName and WaferFileName are changed to LimitFilename and WaferFilename*
Author(s):	Norberto Leite, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- SFET3_L9670_C06_CA_0202 TDF work package shall be loaded (TDF)

Input Specifications

- Content of the SFET3_L9670_C06_CA_0202 directory from the test archive
- Content of the SS_20050623231213_KSI35545_CA0P_N directory from the test archive

Actions Specification

- Copy the SS_20050623231213_KSI35545_CA0P_N into the STDF2EXF Converter's input folder.

```
cd $TEST_CASES_DIR
cp –r SFET3_L9670_C06_CA_0202 $EBS_HOME/data/wpscan/input/TDF
cp –r SS_20050623231213_KSI35545_CA0P_N
$EBS_HOME/data/wpscan/input/STDF
jm –askme;stopjm
```

Output Specifications

- The status of the WP shall be ERROR and the error-message should be: Parameter number should be greater than 0 in case of FCT/ANA parameters.

Post-Conditions:

None

Dependencies

STDF. EBS1096	
Test case identifier:	STDF. EBS1096
Responsibility:	Critical Software
Purpose:	*Automatically generated wafer numbers beginning from 1.*
Author(s):	Norberto Leite, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The Oracle schemas shall be empty (before loading the TDF WP)
- TDF_EZ18A90L_S11P TDF WP shall be loaded

Input Specifications

- Content of the TDF_EZ18A90L_S11P directory from the test archive
- Content of the STDF_TEST_WP_L11, STDF_TEST_WP_L12, STDF_TEST_WP_L13, STDF_TEST_WP_L14, STDF_TEST_WP_L15, STDF_TEST_WP_L16, STDF_TEST_WP_L17, STDF_TEST_WP_L18, STDF_TEST_WP_L19 directories from the test archive

Actions Specification

- Start the JM-Emu if it is not running
- Copy the TDF masterdata into the proper WPScan's input
- Wait until the masterdata are loaded
- Copy the STDF WPs into the STDF2EXF Converter's input

Output Specifications

- The <EBS_ANA>.TRACKING_UNITS table shall contain 9 records, the wafernumber-wafer_id pairs can be

different:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT id, motherlot_id, wafernumber, wafer_id FROM
tracking_units;

  ID MOTHERLOT_ID   WAFERNUMBER WAFER_ID
--------- --------- ----------- ---------
   1 EZ18A90L                 1 K102GKN0
   2 EZ18A90L                 2 K102H2N0
   3 EZ18A90L                 3 K402IGN0
   4 EZ18A90L                 4 KJ02I1N0
   5 EZ18A90L                 5 KN02HFN0
   6 EZ18A90L                 6 KR02HCN0
   7 EZ18A90L                 7 KS02GUN0
   8 EZ18A90L                 8 KT02GTN0
   9 EZ18A90L                 9 KX02H6N0
```

Post-Conditions:

None

Dependencies

STDF. EBS1234	
Test case identifier:	STDF. EBS1234
Responsibility:	Critical Software
Purpose:	*Wrong error handling of inserting MeasZoneVals in STDF2EXF Converter*
Author(s):	Norberto Leite, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The Oracle schemas shall be empty

Input Specifications

- Content of the EXF_L40_L41 directory from the test archive
- Content of the STDF_TEST_WP_L40 directory from the test archive
- Content of the STDF_TEST_WP_L41 directory from the test archive

Actions Specification

- Copy the EXF_L40_L41 WP into the EXF Loader's input and wait until it is loaded
- Stop the jm and copy the two STDF WPs into the STDF2EXF Converter's input
- Start the jm step by step (with the "jm;stopjm" command) until the STDF2EXF Converter starts
- Check the state of the WPs

Output Specifications

- The state of the WPs shall be OK/Done and OK/Delayed
- The number of measurements record in the stage shall be 1

```
sqlplus <ebs_stage>/<ebs_stage>@<conn_string>
SQL> SELECT count(*) FROM measurements;

  COUNT(*)
----------
         1
```

Post-Conditions:

None

Dependencies

STDF. ITMFG9223.1	
Test case identifier:	STDF. ITMFG9223.1
Responsibility:	Critical Software
Purpose:	*Update TESTPROGRAMS.REVISION_DATE field*

Author(s):	Norberto Leite, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The Oracle schemas shall be empty (before loading the TDF WP)
- TDF_TESTPROGRAMS TDF WP shall be loaded
- The flag *EnableMeasTestTypeChecking* in the STDF2EXF Converter's configuration should be set to '*Off*'

Input Specifications

- Content of the STDF_TEST_WP_L30 directory

Actions Specification

- Copy the STDF_TEST_WP_L30 into the STDF2EXF Converter 's input component.

```
cd $TEST_CASES_DIR
cp –r STDF_TEST_WP_L30 $EBS_HOME/data/wpscan/input/STDF
jm –askme
```

Output Specifications

- The <EBS_ANA>.TESTPROGRAMS table shall contain:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT name, revision, to_char(revision_date,'YYYY-MM-DD')
FROM testprograms WHERE revision_date IS NOT NULL;

NAME              REVISION    TO_CHAR(REVISION_DATE,'YYYY-MM
----------------- ----------- ------------------------------
BTS443            1.25        2003-02-27
```

Post-Conditions:

None

Dependencies

STDF.ITMFG9939.2	
Test case identifier:	STDF.ITMFG9939.2
Responsibility:	Critical Software
Purpose:	*Generate DF_ORDER from STDF.TSR*
Author(s):	Norberto Leite, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The Oracle schemas shall be empty (before loading the TDF WP)
- TDF_EZ18A90L_S11P TDF WP shall be loaded

Input Specifications

- Content of the TDF_EZ18A90L_S11P directory from the test archive
- Content of the STDF_TEST_WP_L11 directory from the test archive

Actions Specification

- Get the DF_ORDER values from EBS_ANA:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT p.partp_name, p.pnumber, pa.value FROM
parameter_definitions p, pardef_attribute_values pa WHERE
pa.pardef_id = p.id AND pa.pdfattrtyp_name = 'DF_ORDER' and
p.pnumber IN (1,0,32072);
```

- Copy the WP into the STDF2EXF Converter's input and start jm if it is not running and wait until the WP is loaded
- Get the DF_ORDER values from EBS_ANA:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT p.partp_name, p.pnumber, pa.value FROM
parameter_definitions p, pardef_attribute_values pa WHERE
pa.pardef_id = p.id AND pa.pdfattrtyp_name = 'DF_ORDER' and
```

```
p.pnumber IN (1,0,32072);
```

Output Specifications

- The DF_ORDER values in EBS_ANA after loading the masterrecord shall be:

```
PARTP_NAME           PNUMBER VALUE
---------------- ---------- -------
FCT                   32072 561
HBIN                      1 836
SBIN                      1 871
```

- The DF_ORDER values in EBS_ANA after loading the STDF WP shall be:

```
PARTP_NAME           PNUMBER VALUE
---------------- ---------- -------
ANA                   32072 561
HBIN                      1 836
SBIN                      1 871
SBIN                      0 489
```

Post-Conditions:

None

Dependencies

STDF. ITMFG00011508	
Test case identifier:	STDF. ITMFG00011508
Responsibility:	Critical Software
Purpose:	*Test WP in case of missing Raw file. If the WP doesn't include the raw file an error message shall be logged in the WP log file.*
Author(s):	Norberto Leite, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The jm-emu shall be stopped

Input Specifications

- Content of the MISSING_RAW_FILE work package.

Actions Specification

- Copy the work package into the STDF2EXF Converter's input directory

```
cp –r MISSING_RAW_FILE $EBS_HOME/data/wpscan/input/STDF
```

- Go to the start directory:

```
cd $EBS_HOME/bin
```

- Start jm-emu in single step mode:

```
jm/jm; jm/stopjm
```

- Wait until the STDF2EXF Converter finishes

Output Specifications

- The state and phase of the WP shall be ERROR.
- The following text should be found in the log:

```
Error. Raw file missing
```

Post-Conditions:

None

Dependencies

STDF. ITMFG00013779	
Test case identifier:	STDF. ITMFG00013779
Responsibility:	Critical Software
Purpose:	*Test the parsing of Lot Summary Values.*
Author(s):	Leonardo Fraga, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The jm-emu shall be stopped

Input Specifications

- Content of the ITMFG00013779 directory.

Actions Specification

- Copy the input directory into the STDF2EXF Converter's input directory

```
cp –r ITMFG00013779 $EBS_HOME/data/wpscan/input/STDF
```

- Go to the /bin directory:

```
cd $EBS_HOME/bin
```

- Start jm-emu:

```
jm/jm
```

- Wait until the work package is processed
- Perform the following query on the Analytical schema:

```
SELECT tunit_type, wafernumber FROM tracking_units WHERE
motherlot_id='EL616153';
```

- The result of the query shall be: `WAF, 4`;
- Re-create schemas (cleanDB.sh);
- Update the configuration on the Repository schema:

```
UPDATE configs
SET cf_config = TO_CLOB(UPDATEXML(XMLTYPE(cf_config),
'//LotSummaryValues/text()', 'SUMMARY 004'))
WHERE cf_cfgname = 'Basic Setup'
AND cf_dlcid = (SELECT dc_id
FROM dlcomps
WHERE dc_name='STDF');
COMMIT;
```

- Copy the input directory into the STDF2EXF Converter's input directory.
- Wait until the work packages is processed
- Perform the following query on the Analytical schema: `select tunit_type, lot_id from tracking_units where motherlot_id='EL616153';`

Output Specifications

- The result of the query shall contain:

```
LOT, EL616153Y01.
```

Post-Conditions:

None

Dependencies

STDF.ChipIDLoading	
Test case identifier:	STDF.ITMFG00011698
Responsibility:	Critical Software
Purpose:	*Fill MEASPART_CHIPIDS table*
Author(s):	Hugo Casimiro, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The Oracle schemas shall be empty

Input Specifications

- Content of the STDF_CHIPID_TEST directory from the test archive

Actions Specification

- Copy the WP into the STDF2EXF Converter's input and start jm if it is not running and wait until the WP is loaded
- Run the following command and query:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> spool result.txt
SQL> SELECT * FROM measpart_chipids;
```

This shall create a results.txt file with the result of the SQL query;

Output Specifications

- No differences shall exist between the obtained file (result.txt) and the expected results file *STDF_CHIPID_TEST/MEASPART_CHIPIDS.txt*.
- WP shall end with OK/DONE.

Post-Conditions:

None

Dependncies

STDF.ChipDataAttached	
Test case identifier:	STDF.ITMFG00012868
Responsibility:	Critical Software
Purpose:	*Fill MEASUREMENTS.CHIP_DATA_ATTACHED column*
Author(s):	Hugo Casimiro, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The Oracle schemas shall be empty

Input Specifications

- Content of the STDF_CHIPID_TEST directory from the test archive

Actions Specification

- Copy the WPs into the STDF2EXF Converter's input and start jm if it is not running and wait until the WP is loaded

Output Specifications

- WP shall end with OK/DONE.
- Run the following SQL query:

```
SELECT * FROM measurements;
```

One record shall be returned with the column CHIP_DATA_ATTACHED set to 'Y'.

Post-Conditions:

None

Dependencies

STDF.RawFileSequence	
Test case identifier:	STDF.ITMFG00011698
Responsibility:	Critical Software
Purpose:	*Create 'RAWFILE_SEQ' MeasPartAttribute*
Author(s):	Hugo Casimiro, Daniel Silva

Preconditions

- EBS system shall be installed successfully
- The Oracle schemas shall be empty

Input Specifications

- Content of the STDF_CHIPID_TEST directory from the test archive

Actions Specification

- Copy the WP into the STDF2EXF Converter's input and start jm if it is not running and wait until the WP is loaded
- Run the following command and query:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> spool result.txt
SQL> select * from measpart_attribute where
partattrtp_name='RAWFILE_SEQ';
```

This shall create a results.txt file with the result of the SQL query;

Output Specifications

- No differences shall exist between the obtained file (result.txt) and the expected results file *STDF_RAWFILESEQ_TEST/MEASPART_ATTRIBUTE.txt*.
- WP shall end with OK/DONE.

Post-Conditions:

None

Dependencies

STDF.updateSubMeasStep	
Test case identifier:	ITMFG00014894
Responsibility:	Critical Software
Purpose:	*This test case tests the update of the SubMeasStep value to '0' in case the STDF WP has a 'FCT FE' data source. For all other datasources the SubMeasStep value shall remain unaltered.*
Author(s):	Hugo Casimiro, Daniel Silva

Preconditions

- EBS system shall be installed successfully

Input Specifications

- Both WPs in the STDF_SUBMEASSTEP directory from the input directory

Actions Specification

- Copy the WPs into the STDF2EXF Converter's input
- Start/stop JM until the WPs end processing in the STDF2EXF Converter component.

  ```
  > jm –askme; stopjm
  ```

- Run the following SQL query on EBS_STAGE:

```
select ev.value as wpname, m.id as measID, m.sub_meas_step from MEASUREMENTS m,
ENVIRONMENT_VALUES ev where ev.value like '_E!_WP' escape '!' and
ev.envtp_name='WPName' and ev.meas_id = m.id;
```

- Run JM again and wait until both WPs finish loading.

Output Specifications

- Both WPs shall end with OK/DONE.
- The query on EBS_STAGE shall have returned two rows. In the row for the 'FE_WP' we shall have SUB_MEAS_STEP set to '0'. In the row for the 'BE_WP' we shall have SUB_MEAS_STEP set to '1.

Post-Conditions:

None

Dependencies

Loading parameter checking

The test cases described in this chapter were created for testing the parameter definition checks. They were not developed during the project's period and due to this fact are not included in 4.7 Traceability matrix. Nevertheless, such test cases are necessary to test the converter's integration with the EXF Loader.

STDF.PCHK.01	
Test case identifier:	STDF.PCHK.01
Responsibility:	Critical Software
Purpose:	*HBIN/SBIN parameters have been loaded before loading the STDF.*

Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_01 TDF WP • STDF_TEST_WP_CHK01 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The STDF2EXF Converter shall load the WP successfully** • **The WP-related log file shall contain the line:** `ParmDefs (PARAMETER_DEFINITIONS) records: found 90 inserted 0`	
Post-Conditions:	
None	
Dependencies	

STDF.PCHK.02	
Test case identifier:	STDF.PCHK.02
Responsibility:	Critical Software
Purpose:	*There is no HBIN/SBIN parameter loaded before loading the STDF. The STDF2EXF Converter throws error (InsertNewParamDefs is Off)*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_TESTPROGRAMS TDF WP • STDF_TEST_WP_CHK01 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Set InsertNewParamDefs to Off in the config (STDF, Basic Setup) • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The WP shall be moved into the error** • **The error message shall be similar to:** `ParmDefs (Name: 'BIN_1', ParmNr: '1', ParmType: 'HBIN', MeasAllowedID: '400', TestprgID: '3') was not loaded previously!`	
Post-Conditions:	
None	
Dependencies	

STDF.PCHK.03	
Test case identifier:	STDF.PCHK.03

Responsibility:	Critical Software
Purpose:	*There is no HBIN/SBIN parameter loaded before loading the STDF. The STDF2EXF Converter inserts them.*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_01 TDF WP • STDF_TEST_WP_CHK01 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Set InsertNewParamDefs to On in the config (STDF2EXF Converter, Basic Setup) • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The STDF2EXF Converter shall load the WP successfully** • **The WP-related log file shall contain the line:** `ParmDefs (PARAMETER_DEFINITIONS) records: found 90 inserted 0`	
Post-Conditions:	
None	
Dependencies	

STDF.PCHK.04	
Test case identifier:	STDF.PCHK.04
Responsibility:	Critical Software
Purpose:	*FCT/ANA parameters have been loaded before loading the STDF. The STDF file is "test by number" type. (parmNr field is filled).*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_04 TDF WP • STDF_TEST_WP_CHK04 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The STDF2EXF Converter shall load the WP successfully** • **The WP-related log file shall contain the line:** `ParmDefs (PARAMETER_DEFINITIONS) records: found 90 inserted 0`	
Post-Conditions:	
None	
Dependencies	

STDF.PCHK.05	
Test case identifier:	STDF.PCHK.05
Responsibility:	Critical Software
Purpose:	*There is no FCT/ANA parameter loaded before loading the STDF.* *The STDF file is "test by number" type (parmNr field is filled) and the STDF2EXF Converter throws error (InsertNewParamDefs is Off).*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_05TDF WP • STDF_TEST_WP_CHK05 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The WP shall finish with error.** • **The error message shall be similar to:** **`ParmDefs (Name: 'IPD_TEST', ParmNr: '4010', ParmType: 'ANA', MeasAllowedID: '200', TestprgID: '1') was not loaded previously!`**	
Post-Conditions:	
None	
Dependencies	

STDF.PCHK.06	
Test case identifier:	STDF.PCHK.06
Responsibility:	Critical Software
Purpose:	*There is no FCT/ANA parameter loaded before loading the STDF.* *The STDF file is "test by number" type (parmNr field is filled) and the STDF2EXF Converter inserts the missing parameters (InsertNewParamDefs is On).*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_05TDF WP • STDF_TEST_WP_CHK04 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The STDF2EXF Converter shall load the WP successfully** • **The WP-related log file shall contain the line:** **`ParmDefs (PARAMETER_DEFINITIONS) records: found 42 inserted 148`**	
Post-Conditions:	
None	

Dependencies

STDF.PCHK.07	
Test case identifier:	STDF.PCHK.07
Responsibility:	Critical Software
Purpose:	*FCT/ANA parameters have been loaded before loading the STDF. The STDF file is "test by number" type. (parmNr field is filled). The STDF2EXF Converter finds the parameters through the parameter name and not the number.*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_07TDF WP • STDF_TEST_WP_CHK04 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The STDF2EXF Converter shall load the WP successfully** • **The WP-related log file shall contain the line:** **`ParmDefs (PARAMETER_DEFINITIONS) records: found 190 inserted 0`**	
Post-Conditions:	
None	
Dependencies	

STDF.PCHK.08	
Test case identifier:	STDF.PCHK.08
Responsibility:	Critical Software
Purpose:	*FCT/ANA parameters have been loaded before loading the STDF. The STDF file is "test by number" type. (parmNr field is filled). The STDF2EXF Converter finds the parameters through the parameter number, but the parameter type shall be updated (FCT=>ANA, ANA=>FCT).*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_08TDF WP • STDF_TEST_WP_CHK04 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	

- **The WP shall be loaded successfully**
- **The WP-related log file shall contain the line:**
  `ParmDefs (PARAMETER_DEFINITIONS) records: found 190 inserted 0`
- **The <EBS_ANA>. PARAMETER_DEFINITIONS table shall contain:**

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT pnumber,partp_name,name FROM parameter_definitions
WHERE partp_name NOT LIKE '%BIN' AND pnumber IN
(1010,3010,3200,3464,4010,4031,10000) ORDER BY pnumber;

   PNUMBER PARTP_NAME   NAME
---------- ------------ ----------------
      1010 FCT          CONT_SHORT_TEST
      3010 FCT          SCAN_CHECK_SPO
      3200 FCT          BSCAN_FUNC
      3464 FCT          BSCAN_MAX_BS
      4010 ANA          IPD_TEST
      4031 ANA          IPDS_TEST
     10000 FCT          BIST_TEST_FUNC
```

Post-Conditions:

None

Dependencies

STDF.PCHK.09	
Test case identifier:	STDF.PCHK.09
Responsibility:	Critical Software
Purpose:	*FCT/ANA parameters have been loaded before loading the STDF.* *The STDF file is "test by number" type. (parmNr field is filled)* *The STDF2EXF Converter finds the parameters through the parameter name and the parameter type shall be updated (FCT=>ANA, ANA=>FCT).*
Author(s):	Norberto Leite, Daniel Silva

Preconditions

- EBS system shall be installed successfully.

Input Specifications

- TDF_PCHK_09 TDF WP
- STDF_TEST_WP_CHK01 STDF WP

Actions Specification

- Stop the JM and wait until all components exist
- Re-create the Staging Area and Analytical (and Operational) Schemas and packages
- Set InsertNewParamDefs to On in the config (STDF2EXF Converter, Basic Setup)
- Start the JM
- Copy the TDF master WP into the proper wpscan's input
- Wait until the masterdata are loaded
- Copy the STDF WPs into the STDF2EXF Converter's input

Output Specifications

- **The WP shall be loaded successfully**
- **The WP-related log file shall contain line:**
  `ParmDefs (PARAMETER_DEFINITIONS) records: found 190 inserted 0`
- **The <EBS_ANA>. PARAMETER_DEFINITIONS table shall contain:**

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT pnumber,partp_name,name FROM parameter_definitions
WHERE partp_name NOT LIKE '%BIN' AND name IN
('BIST_TEST_FUNC','BSCAN_FUNC','BSCAN_MAX_BS',
'CONT_SHORT_TEST','IPDS_TEST','IPD_TEST',
'SCAN_CHECK_SPO') ORDER BY name;
```

```
   PNUMBER PARTP_NAME   NAME
---------- ------------ ---------------
           FCT          BIST_TEST_FUNC
           FCT          BSCAN_FUNC
           FCT          BSCAN_MAX_BS
           FCT          CONT_SHORT_TEST
           ANA          IPDS_TEST
           ANA          IPD_TEST
           FCT          SCAN_CHECK_SPO
```

Post-Conditions:

None

Dependencies

STDF.PCHK.10	
Test case identifier:	STDF.PCHK.10
Responsibility:	Critical Software
Purpose:	*FCT/ANA parameters have been loaded before loading the STDF.* *The STDF file is "test by name" type. (parmNr field is NULL)*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_10 TDF WP • STDF_TEST_WP_CHK10 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Set InsertNewParamDefs to On in the config (STDF, Basic Setup) • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The STDF2EXF Converter shall load the WP successfully** • **The WP-related log file shall contain the line:** **`ParmDefs (PARAMETER_DEFINITIONS) records: found 116 inserted 0`**	
Post-Conditions:	
None	
Dependencies	

STDF.PCHK.11	
Test case identifier:	STDF.PCHK.11
Responsibility:	Critical Software
Purpose:	*There is no FCT/ANA parameter loaded before loading the STDF.* *The STDF file is "test by name" type (parmNr field is NULL) and the STDF2EXF Converter throws error (InsertNewParamDefs is Off).*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	

• TDF_PCHK_11 TDF WP • STDF_TEST_WP_CHK10 STDF WP
Actions Specification
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Set InsertNewParamDefs to On in the config (STDF, Basic Setup) • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input
Output Specifications
• **The WP shall finish with ERROR** • **The error message shall be similar to:** **`ParmDefs (Name: 'B_N01__25U', ParmNr: '', ParmType: 'ANA', MeasAllowedID: '600', TestprgID: '1') was not loaded previously!`**
Post-Conditions:
None
Dependencies

STDF.PCHK.12	
Test case identifier:	STDF.PCHK.12
Responsibility:	Critical Software
Purpose:	*There is no FCT/ANA parameter loaded before loading the STDF.* *The STDF file is "test by name" type (parmNr field is NULL) and the STDF2EXF Converter inserts the missing parameters (InsertNewParamDefs is On).*
Author(s):	Norberto Leite, Daniel Silva
Preconditions	
• EBS system shall be installed successfully.	
Input Specifications	
• TDF_PCHK_11 TDF WP • STDF_TEST_WP_CHK10 STDF WP	
Actions Specification	
• Stop the JM and wait until all components exist • Re-create the Staging Area and Analytical (and Operational) Schemas and packages • Set InsertNewParamDefs to On in the config (STDF, Basic Setup) • Start the JM • Copy the TDF master WP into the proper wpscan's input • Wait until the masterdata are loaded • Copy the STDF WPs into the STDF2EXF Converter's input	
Output Specifications	
• **The STDF2EXF Converter shall load the WP successfully** • **The WP-related log file shall contain line:** **`ParmDefs (PARAMETER_DEFINITIONS) records: found 114 inserted 2`**	
Post-Conditions:	
None	
Dependencies	

STDF.PCHK.13	
Test case identifier:	STDF.PCHK.13
Responsibility:	Critical Software

Purpose:	*FCT/ANA parameters have been loaded before loading the STDF. The STDF file is "test by name" type. (parmNr field is NULL). The STDF2EXF Converter finds the parameters through the parameter name and the parameter type shall be updated (FCT=>ANA).*
Author(s):	Norberto Leite

Preconditions

- EBS system shall be installed successfully.

Input Specifications

- TDF_PCHK_13 TDF WP
- STDF_TEST_WP_CHK10 STDF WP

Actions Specification

- Stop the JM and wait until all components exist
- Re-create the Staging Area and Analytical (and Operational) Schemas and packages
- Start the JM
- Copy the TDF master WP into the proper wpscan's input
- Wait until the masterdata are loaded
- Copy the STDF WPs into the STDF2EXF Converter's input

Output Specifications

- **The WP shall be loaded successfully**
- **The WP-related log file shall contain line:**
  ParmDefs (PARAMETER_DEFINITIONS) records: found 116 inserted 0
- **The <EBS_ANA>. PARAMETER_DEFINITIONS table shall contain:**

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT pnumber,partp_name,name FROM parameter_definitions
WHERE name LIKE 'M%' ORDER BY name;

  PNUMBER PARTP_NAME   NAME
--------- ------------ ---------------
          ANA          MB_N02__C
          ANA          MB_P01__C
          ANA          MB_P28.L
          ANA          MIC_N02
          ANA          MIC_N08
          ANA          MIC_P01
          ANA          MIC_P28.L
          ANA          MR_SPH.A/B__C
```

Post-Conditions:

None

Dependencies

STDF.PCHK.14	
Test case identifier:	STDF.PCHK.14
Responsibility:	Critical Software
Purpose:	*If the parameter is only in TSR, the parameter type in the database shall be updated only if the new type is ANA and the old one (in the database) is FCT. The STDF file is "test by number" type. (parmNr field is filled)*
Author(s):	Norberto Leite

Preconditions

- EBS system shall be installed successfully.

Input Specifications

- TDF_PCHK_2x TDF WP
- STDF_TEST_WP_CHK20 STDF WP
- STDF_TEST_WP_CHK21 STDF WP

Actions Specification

- Stop the JM and wait until all components exist

- Re-create the Staging Area and Analytical (and Operational) Schemas and packages
- Start the JM
- Copy the TDF master WP into the proper wpscan's input
- Wait until the masterdata are loaded
- Copy STDF_TEST_WP_CHK20 WP into the STDF2EXF Converter's input
- Wait until it is loaded and execute the query (see below)
- Copy STDF_TEST_WP_CHK21 WP into the STDF2EXF Converter's input
- Wait until it is loaded and execute the query (see below)
- Copy STDF_TEST_WP_CHK20 WP into the STDF2EXF Converter's input
- Wait until it is loaded and execute the query (see below)

Output Specifications

- **The WPs shall be loaded successfully**
- **The <EBS_ANA>. PARAMETER_DEFINITIONS table shall contain after the first loading:**
- **sqlplus <ebs_ana>/<ebs_ana>@<conn_string>**

```
SQL> SELECT pnumber,partp_name FROM parameter_definitions WHERE
pnumber=7 AND partp_name NOT LIKE '%BIN';

  PNUMBER PARTP_NAME
--------- ------------
        7  FCT
```

- **The <EBS_ANA>. PARAMETER_DEFINITIONS table shall contain after the second and third loadings:**

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT pnumber,partp_name FROM parameter_definitions WHERE
pnumber=7 AND partp_name NOT LIKE '%BIN';

  PNUMBER PARTP_NAME
--------- ------------
        7  ANA
```

Post-Conditions:

None

Dependencies

STDF.PCHK.15	
Test case identifier:	STDF.PCHK.15
Responsibility:	Critical Software
Purpose:	*If the parameter is only in TSR, the parameter type in the database shall be updated only if the new type is ANA and the old one (in the database) is FCT. The STDF file is "test by number" type. (parmNr field is filled)*
Author(s):	Norberto Leite

Preconditions

- EBS system shall be installed successfully.

Input Specifications

- TDF_PCHK_3x TDF WP
- STDF_TEST_WP_CHK30 STDF WP
- STDF_TEST_WP_CHK31 STDF WP

Actions Specification

- Stop the JM and wait until all components exist
- Re-create the Staging Area and Analytical (and Operational) Schemas and packages
- Start the JM
- Copy the TDF master WP into the proper wpscan's input
- Wait until the masterdata are loaded
- Copy STDF_TEST_WP_CHK30 WP into the STDF2EXF Converter's input
- Wait until it is loaded and execute the query (see below)

- Copy STDF_TEST_WP_CHK31 WP into the STDF2EXF Converter's input
- Wait until it is loaded and execute the query (see below)
- Copy STDF_TEST_WP_CHK30 WP into the STDF2EXF Converter's input
- Wait until it is loaded and execute the query (see below)

Output Specifications

- The WPs shall be loaded successfully
- The <EBS_ANA>. PARAMETER_DEFINITIONS table shall contain after the first loading:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT pnumber,partp_name,name FROM parameter_definitions
WHERE name='AAAA_N_.6/1.3_5V';

  PNUMBER PARTP_NAME    NAME
--------- ------------ ------
          FCT          AAAA_N_.6/1.3_5V
```

- The <EBS_ANA>. PARAMETER_DEFINITIONS table shall contain after the second and third loadings:

```
sqlplus <ebs_ana>/<ebs_ana>@<conn_string>
SQL> SELECT pnumber,partp_name,name FROM parameter_definitions
WHERE name='AAAA_N_.6/1.3_5V';

  PNUMBER PARTP_NAME    NAME
--------- ------------ ------
          ANA          AAAA_N_.6/1.3_5V
```

Post-Conditions:

None

Dependencies

STDF.PCHK.16	
Test case identifier:	STDF.PCHK.16
Responsibility:	Critical Software
Purpose:	*Do not update transaction_id when updating parameter type and adaptation SQL statements in STDF2EXF Converter's parameter update*
Author(s):	Norberto Leite

Preconditions

- EBS system should be installed successfully
- Oracle schemas (stage, ana and oper, if it exists) shall be empty
- JM shall be running.

Input Specifications

- TDF_PCHK_2x, TDF_PCHK_3x TDF WPs
- STDF_TEST_WP_CHK21, STDF_TEST_WP_CHK31 STDF WPs

Actions Specification

- Set LoaderSQL to 1 in the config (STDF, Logging)
- Copy the TDF WPs into the input and wait until they are loaded
- Stop the JM and update the staging area:

```
sqlplus <ebs_stage>/<ebs_stage>@<conn_string>
SQL> UPDATE parameter_definitions SET partp_name='FCT',
recordstate='MASTERRECORD', transaction_id = 1 WHERE partp_name
IN ('ANA','FCT') AND (name='AAAA_N_.6/1.3_5V' OR pnumber=7);
SQL> commit;
```

- Copy the STDF_TEST_WP_CHK21 and STDF_TEST_WP_CHK31 WPs into the STDF2EXF Converter's input and start the jm step by step until the STDF2EXF Converter instances start processing that WPs
- Get the parameter types from the staging area:

```
sqlplus <ebs_stage>/<ebs_stage>@<conn_string>
SQL> SELECT partp_name, pnumber, name, transaction_id,
recordstate FROM parameter_definitions WHERE partp_name IN
```

```
('ANA','FCT') AND (name='AAAA_N_.6/1.3_5V' OR pnumber=7) ORDER
BY name;
PARTP_NAME PNUMBER NAME            TRANSACTION_ID RECORDSTATE
---------- -------- --------------- -------------- -----------
ANA                 AAAA_N_.6/1.3_5V              0 MASTERRECORD
ANA               7                               0 MASTERRECORD
SQL> UPDATE parameter_definitions SET recordstate='OK',
transaction_id=0;
SQL> UPDATE parameter_definitions SET partp_name='FCT' WHERE
partp_name IN ('ANA','FCT') AND (name='AAAA_N_.6/1.3_5V' OR
pnumber=7);
SQL> commit;
```

- Load these WPs, stop the jm and copy the STDF_TEST_WP_CHK21 and STDF_TEST_WP_CHK31 WPs into the STDF2EXF Converter's input and start the jm step by step until the STDF2EXF Converter instances start processing that WPs
- Get the parameter types from the staging area:

```
sqlplus <ebs_stage>/<ebs_stage>@<conn_string>
SQL> SELECT partp_name, pnumber, name, transaction_id,
recordstate FROM parameter_definitions WHERE partp_name IN
('ANA','FCT') AND (name='AAAA_N_.6/1.3_5V' OR pnumber=7) ORDER
BY name;
PARTP_NAME PNUMBER NAME            TRANSACTION_ID RECORDSTATE
---------- -------- --------------- -------------- -----------
ANA                 AAAA_N_.6/1.3_5V              0 Aggr
ANA               7                               0 Aggr
```

- Wait until all of the STDF2EXF Converter instances exit and check the latest component-related log file:

```
bash-2.05$ cd $EBS_HOME/data/loader/STDF/log
bash-2.05$ cat stdf_*.log | grep "PARAMETER_DEFINITIONS" | grep
-e "[n|N][v|V][l|L]"
bash-2.05$
```

Output Specifications

- The queries shall return the same result as above

Post-Conditions:

None

Dependencies

STDF.PCHK.17	
Test case identifier:	STDF.PCHK.17
Responsibility:	Critical Software
Purpose:	*Oracle deadlocks because of update on par_def. The STDF2EXF Converter shall update the parameter definitions table in order of the primary key*
Author(s):	Norberto Leite

Preconditions

- EBS system should be installed successfully
- Oracle schemas (stage, ana and oper, if it exists) shall be empty
- JM shall be running.

Input Specifications

- TDF_TESTPROGRAMS TDF WP
- STDF_TEST_WP_07, STDF_TEST_WP_08 STDF WPs

Actions Specification

- Copy the TDF WPs into the input and wait until they are loaded
- Copy the STDF WPs into the input and wait until they are loaded
- Update the staging area:

```
sqlplus <ebs_stage>/<ebs_stage>@<conn_string>
SQL> UPDATE parameter_definitions SET partp_name = decode (
partp_name,'ANA','FCT','ANA')WHERE partp_name IN('ANA','FCT');
```

`SQL> commit;`
• Copy the STDF WPs into the input and wait until they are loaded
Output Specifications

• The wp-related log-file of STDF_TEST_WP_07 WP shall contain:
`156 existing parameter definitions were updated to ANA type`
• The wp-related log-file of STDF_TEST_WP_08 WP shall contain:
`69 existing parameter definitions were updated to ANA type`

Post-Conditions:
None
Dependencies

STDF.PCHK.18	
Test case identifier:	ITMFG00011957
Responsibility:	Critical Software
Purpose:	*For the field STDF4.Wir.wafer_id no control characters are allowed (hex 01 til 1F and 7F). If some of this invalid chars are found the WP should end with error.*
Author(s):	Norberto Leite
Preconditions	
• EBS system should be installed successfully • JM shall be stopped.	
Input Specifications	
• INVALID_CONTROL_CHARS WP	
Actions Specification	

• Copy the INVALID_CONTROL_CHARS WPs into the input of the STDF loader
• Start the jm-emulator
`cp –r INVALID_CONTROL_CHARS $EBS_HOME/data/loader/STDF/input`
`jm –askme;`
• The WP should end with error and placed in the error folder.	

Output Specifications	
• The wp-related log-file of INVALID_CONTROL_CHARS shall contain: `(STDF) - (ERROR ) - Error occurred, invalid control character:`	
Post-Conditions:	
None	
Dependencies	

Rosetta Net support

The test cases shown below should test the integration of the loaded information from STDF files with an external component of EBS – the Rosetta Net Converter. Such test cases were not developed during the project's period and due to this fact are not included in 4.7 Traceability matrix. Nevertheless, such test cases are necessary to test the converter's integration with the EXF Loader.

STDF.ITMFG14696.1	
Test case identifier:	STDF. ITMFG14696.1
Responsibility:	Critical Software

Purpose:	*This test case test the.mapping of several STDF fields into the ENVIRONMENT_VALUES and PARDEF_ATTRIBUTE_VALUES tables, related to the Rosetta Net STDF files.*
Author(s):	Hugo Casimiro, Daniel Silva

Preconditions

- EBS system should be installed successfully
- DB schemas should be clean

Input Specifications

STDFLoader.inp\STDF\STDF_TEST_WP_ROS2

Actions Specification

- Copy the WP directory into /data/loader/STDF/input directory;
- Start JM;
- Wait until WP is fully loaded and run the following queries on EBS_ANA:

```
1. select envtp_name, value from environment_values where
meas_id = (select meas_id from environment_values e where
e.value='STDF_TEST_WP_ROS2') order by envtp_name;

2. select * from pardef_attribute_values where
pdfattrtyp_name='TEST_NO' or pdfattrtyp_name='CATEGORY';
```

Output Specifications

- The WP should have ended with status OK/Done
- After running the first query you should have a list of environment types and corresponding values. The following should be part of the list:

```
ENVTP_NAME              VALUE
-----------------------------------------------------------------
ConverterVersion        c_st3_XML_version 1.0.0
CustomerPN              0000032N5247
FileCreationDate        2007-04-13T19:53:23Z
FlatNotchLocation       270
OriginLocation          45
TestLocation            BTV
WaferDiameter           200
XIsHorizontal           Y
```

- The result of the second SQL query should be a list of ParDefAttributeValues for type ‘CATEGORY’ only. No results should exist for type ‘TEST_NO’.

Post-Conditions:

None

Dependencies

STDF.ITMFG14696.2	
Test case identifier:	STDF. ITMFG14696.2
Responsibility:	Critical Software
Purpose:	*This test case test the.mapping of ParDef_Attribute_Values for type ‘TEST_NO’*
Author(s):	Hugo Casimiro

Preconditions

- EBS system should be installed successfully

Input Specifications

STDFLoader.inp\STDF\STDF_TEST_WP_ROS1

Actions Specification

- copy the WP directory into /data/loader/STDF/input directory;
- Start JM;
- Wait until WP is fully loaded and run the following queries on EBS_ANA:

```
1. select envtp_name, value from environment_values where
```

```
meas_id = (select meas_id from environment_values e where
e.value='STDF_TEST_WP_ROS1') and envtp_name='ConverterVersion';

2. select * from pardef_attribute_values where
pdfattrtyp_name='TEST_NO';
```

Output Specifications

- The WP should have ended with status OK/Done
- After running the first query you should have the following Environment value:

```
ENVTP_NAME              VALUE
-------------------------------------------------------------
ConverterVersion        c_st4_XML_version 1.0.0
```

- The result of the second SQL query should be a list of ParDefAttributeValues for type 'TEST_NO'.

Post-Conditions:

None

Dependencies

4.7 Traceability matrix

A traceability matrix provides an easy way to relate test cases to requirements or use cases, providing developers a good insight of which tests should be executed for a specific purpose. For any use case or requirement it is possible to determine which test or tests shall be executed in order to validate what was described in such items. Whenever a cross connects a line of the matrix with a column, it means that a relationship is present between the line and column, thus between a use case or requirement and a test case, as seen in the following matrixes presented below.

Test cases to requirements matrix

Figure 20 shows the traceability matrix between test cases and requirements for the STDF2EXF Converter.

	Functional Requirements::STDF2EXF-SRS-FUN-001 Data conversion	Functional Requirements::STDF2EXF-SRS-FUN-002 Working modes	Functional Requirements::STDF2EXF-SRS-FUN-003 Configuration	Functional Requirements::STDF2EXF-SRS-FUN-004 Logging	Functional Requirements::STDF2EXF-SRS-FUN-006 Converter help	Implementation::STDF2EXF-SRS-IMP-001 STDF Converter structure	Implementation::STDF2EXF-SRS-IMP-002 File naming convention	Interoperability::STDF2EXF-SRS-INT-001 Integration with EBS	Resources::STDF2EXF-SRS-RES-001 Operating system	Validation::STDF2EXF-SRS-VAL-001 STDF validation	Validation::STDF2EXF-SRS-VAL-002 XML structure validation	Validation::STDF2EXF-SRS-VAL-003 XML data integrity
Configuration::STDF2EXF-TCS-CFG-002 INI file recognition (fail case)			X									
Conversion::STDF2EXF-TCS-CON-001 Data conversion	X						X	X	X			
Help::STDF2EXF-TCS-HLP-001 Help					X							
Logging::STDF2EXF-TCS-LOG-001 Logging				X								
Structure::STDF2EXF-TCS-STR-001 Structure						X						
Validation::STDF2EXF-TCS-VAL-001 STDF recognition (success case)										X		
Validation::STDF2EXF-TCS-VAL-002 STDF recognition (fail case)										X		
Validation::STDF2EXF-TCS-VAL-003 XML structure validation											X	
Validation::STDF2EXF-TCS-VAL-004 XML data integrity												X
Validation::STDF2EXF-TCS-VAL-005 Raw files validation (success case)										X		
Validation::STDF2EXF-TCS-VAL-006 Raw files validation (fail case)										X		
Validation::STDF2EXF-TCS-VAL-007 Work package validation (success case)	X											
Validation::STDF2EXF-TCS-VAL-008 Work package validation (fail case)	X											
Working modes::STDF2EXF-TCS-WRK-001 Standalone mode		X										

Figure 20: Test cases to requirements matrix

As seen above, each functional requirement has at least one test case associated with it. This guarantees that every requirement will be tested somehow, assuring that the application to be developed behaves as specified and as expected. Some requirements might, however, have more than one test case associated with them, as it happens with `STDF2EXF-SRS-VAL-001`, which has four correspondent test cases.

Use cases to test cases matrix

Figure 21 shows the traceability matrix between use cases and test cases for the STDF2EXF Converter.

	Configuration::STDF2EXF-TCS-CFG-001 INI file recognition (success case)	Configuration::STDF2EXF-TCS-CFG-002 INI file recognition (fail case)	Conversion::STDF2EXF-TCS-CON-001 Data conversion	Help::STDF2EXF-TCS-HLP-001 Help	Logging::STDF2EXF-TCS-LOG-001 Logging	Structure::STDF2EXF-TCS-STR-001 Structure	Validation::STDF2EXF-TCS-VAL-001 STDF recognition (success case)	Validation::STDF2EXF-TCS-VAL-002 STDF recognition (fail case)	Validation::STDF2EXF-TCS-VAL-003 XML structure validation	Validation::STDF2EXF-TCS-VAL-004 XML data integrity	Validation::STDF2EXF-TCS-VAL-005 Raw files validation (success case)	Validation::STDF2EXF-TCS-VAL-006 Raw files validation (fail case)	Validation::STDF2EXF-TCS-VAL-007 Work package validation (success case)	Validation::STDF2EXF-TCS-VAL-008 Work package validation (fail case)	Working modes::STDF2EXF-TCS-WRK-001 Standalone mode
Use Cases::STDF2EXF-SRS-USR-010 Start the converter															
Use Cases::STDF2EXF-SRS-USR-011 Start the converter in normal mode	X	X	X		X	X	X	X	X	X	X	X			
Use Cases::STDF2EXF-SRS-USR-012 Start the converter in standalone mode	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Use Cases::STDF2EXF-SRS-USR-030 Manage the converter's files	X	X													
Use Cases::STDF2EXF-SRS-USR-040 Configure the converter	X	X													

Figure 21: Use cases to test cases matrix

Analogously to the test cases to requirements matrix, each use case has one or more test cases associated with it, guaranteeing that each use case is properly tested. Please note the use case `STDF2EXF-SRS-USR-010` is a super use case which includes the sub use cases `STDF2EXF-SRS-USR-011` and `STDF2EXF-SRS-USR-012`.

5 Prototype development

A vertical prototype for the STDF2EXF Converter was built, based on the requirements that were gathered for this new component, and later tested according to its test case specification.

It was developed with the Eclipse Platform using the programming language C++, together with PL/SQL and XML. There were not alternative technologies to be considered for development purposes, since the new component would be part of an existing complex system that was developed with such languages. The trainee had no specific experience with C++, nevertheless, had made previous projects using C together with object oriented programming languages such as C# or Java. The trainee had as well previous experience with database programming, using SQL, and a good knowledge of the XML format.

All development was made on top of the latest stable version of EBS DB available at Critical Software's CVS (Concurrent Versions System) by the beginning of the project's Design Engineering Phase. The prototype was built based on EBS DB v2.3.0.3 and it was intended to support the functionalities provided in this version.

This prototype aims to show its working mode beyond a basic level, as if the user would be using a released version. In fact, when working in standalone mode, the STDF2EXF Converter prototype is fully functional, meaning that it can be used with no limitations, even if not running over any EBS DB installation whatsoever. When in this mode, the converter processes STDF work packages from an input path selected by the user and transforms its raw files into XML based EXF files. As a result, a prototype of this level allows one to have a clear idea of what the developed application should do and how it should behave. This should allow the client to decide with a high percentage of confidence whether or not this converter, when introduced in EBS DB, could indeed be a real advantage in comparison to the currently used solution.

5.1 Prototype engineering

The business know-how gained during the first week of this project was an essential step towards the understanding of how the EBS Data Loading components are structured and connected, due to the system's high complexity and the fact that various development teams have embraced it since its creation.

The goals of the project concerning the prototype development were clear, to design a new STDF2EXF converter that could work both integrated in EBS and independently from it. This converter would incorporate the logic regarding the conversion process of the existing STDF Loader, being integrated at a later stage with the EXF Loader, the component that loads the information processed by the converters into a staging database.

Most of the conversion processes are shared by the different converters, and this lead to the creation of coding templates that can and should be used when a new converter needs to be developed. Although, in general terms, these processes are identical independently of the converter type to be used, it is not less true that the conversion logic itself changes considerably depending on the file type to be converted. Each file type which is extracted from the factory testers is processed by EBS DB differently according to its characteristics. It is common that the information of such files is divided into zones which are organised

differently according to their zone type, e.g., header or data. Among zones of the same type in different file formats, the information can be handled differently as well, e.g., *setsign* or *lookup*. These differences, as one can see, added to the different requirements of each converter and their inherent logic, might lead to quite significant variations in the way EBS Data Loading converters are designed and implemented.

Although what was mentioned above is valid for text based raw file formats, such as 2DPATREC or APRC, it is not true for binary files like STDF. The existing converter templates were designed according to the structural needs of the different text type based converters, which are in various ways different than binary type based converters. The STDF2EXF Converter would, in this way, share a significant percentage of the structure built for the existing converters, but also have in some extent nuances that one might notice as structurally different from the standard text based converters that are part of EBS DB.

The existing STDF Loader combines the structure and the logic of existing converters and the EXF Loader, making it a unique component in the EBS Data Loading scenario. Not only the STDF Loader was implemented with some of the conversion mechanisms that exist in other converters but it also has common steps of the loading process obtained from the EXF Loader. However, such mechanisms have their uniqueness since they process binary data, significantly different from text data, as mentioned before. Added to this, a major difference between the processing steps made by the STDF Loader and the converters plus EXF Loader resides in the fact that, while the latter transform raw data files into EXF files and load the generated information into the system's staging database, the former skips EXF file creation and converts the files in memory, loading their data afterwards into the database by itself.

In this way, a compromise between the standard converters' structure and the STDF Loader's approach to the handling of STDF files had to be made. This was done in order to create a new converter that would incorporate the properties a converter must have in order to act as one, but at the same time being able to handle STDF files in an autonomous and practical way.

The standalone mode

Besides acting as a standard converter that would transform STDF raw files into XML files that would be later loaded into the staging area of EBS DB, this application should as well have the capability of being completely independent from EBS DB, something which is not shared by the system's other converters. Each component of the EBS Data Loading retrieves and stores data from and into the repository database, which contains configurations and information that are essential to the execution of every component that runs in EBS DB.

The Data Loading mechanism is controlled by the JM Emulator, a process that, supported by the repository database, manages the EBS DB components by launching them and sending them instructions so that these can carry out their tasks. Each component is registered in the EBS_REPO database, which stores core information to the EBS Data Loading such as running processes and their states, the streams associated to the different components, the workflows that must be followed, paths for the components' logfiles and binaries, the work packages and raw files that are present in the system, their status and other data which the system is supported by in order to run properly.

The fact that each component is so intimately connected with the system's workflow and the repository database made the process of creating such a unique converter as something

interesting and challenging. In order to achieve it, the configurations used by the new converter were copied from the database to .ini files that could be accessed locally by the application, if in standalone mode. When running in this mode, the application does not require a single connection to any database, making use of information that can be accessed locally, either through the code itself or by retrieving information from such configuration files. This allows the application to be added, for instance, to a compressed file such as .zip, .rar or other, and to be extracted by any user to his HP-UX machine, being ready to use without any installation procedure whatsoever. On the other hand, the application when not running in standalone mode acts as a standard converter, integrated in the EBS workflow and transforming input raw STDF files into XML based EXF files which are then forwarded to the next component in the workflow as defined in the repository database.

The STDF2EXF Converter can be initialized in two distinct ways: by the user or by the EBS system – JM Emulator. The former is valid when the program is to be launched in standalone mode, while the latter happens when the converter is used in its normal environment, integrated with EBS. The principle behind the standalone mode is that a user can call the program through the command line using the path of a STDF work package to be converted as argument, preceded by the flag `-s` (Figure 22). The application will simply convert the binary information of the STDF raw file contained inside the work package into an EXF file placed inside this same folder.

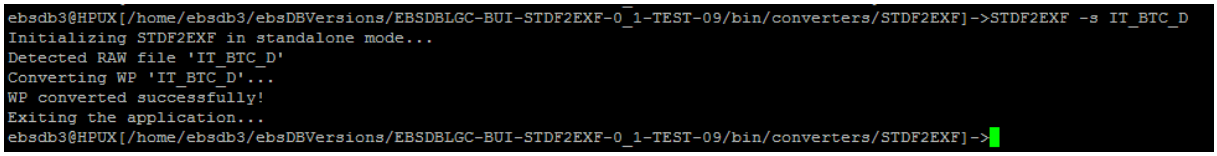

Figure 22: STDF2EXF Converter working in standalone mode

As seen above, the process of converting a STDF work package in standalone mode is quite straightforward. The user invokes the required command line and the application quickly converts the data, exiting afterwards.

Figure 23 and Figure 24 show examples of two generated files from the STDF2EXF Converter: firstly, a log generated after the conversion of a STDF work package; secondly, a EXF file generated from a STDF raw file. Such examples are valid for the STDF Loader as well and they should be identical.

```
Fri Mar  7 11:10:38 2008 - (STDF) - (INFO      ) - Start converting 'IT_BTC_D' RAW file
Fri Mar  7 11:10:40 2008 - (STDF) - (INFO      ) - Start processing PRR records, collecting
MeasPartAttrs elements
Fri Mar  7 11:10:40 2008 - (STDF) - (INFO      ) - Start processing PTR/FTR records,
collecting MeasPartVals elements
Fri Mar  7 11:10:40 2008 - (STDF) - (INFO      ) - Datalog sequence is serialized
Fri Mar  7 11:10:42 2008 - (STDF) - (WARNING   ) - Warning. 4 PTR records were ignored,
because they were found before the first PIR record
Fri Mar  7 11:10:42 2008 - (STDF) - (INFO      ) - Start processing ParmAggrs records
Fri Mar  7 11:10:42 2008 - (STDF) - (INFO      ) - Writing out MeasPartPos, MeasPartChipIDs,
MeasPartAttrs and MeasPartVals records
Fri Mar  7 11:10:43 2008 - (STDF) - (INFO      ) - 'IT_BTC_D' RAW-file converted in 5.20
seconds
Fri Mar  7 11:10:43 2008 - (STDF) - (INFO      ) - 'IT_BTC_D' WP converted SUCCESSFULLY
Fri Mar  7 11:10:43 2008 - (STDF) - (INFO      ) - 'IT_BTC_D' WP converted in 5.21 seconds
```

Figure 23: STDF work package log after conversion in standalone mode

```
<?xml version="1.0" encoding="UTF-8"?>
<EXF xmlns="http://www.infineon.com/exf"
xmlns:exf2db="http://www.infineon.com/exf2db"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.infineon.com/exf /home/ebsdb3/ebsDBVersions/EBSDBLGC-BUI-
STDF2EXF-0_1-TEST-09/EXF_schema/EXF_Schema_qualified.xsd"
Version="V02.7">
<MeasAllowed ID="1" MeasStep="S3" MeasDataSource="FCT FE" SumDesc="NA"/>
<TrackingUnit ID="2" TUType="WAF" CreateDate="2008-03-07T11:10:40" WafNr="001"
MotherLotID="02HN0P78"/>
<MeasZoneVals ID="3" ZoneType="NO_ZONE" Val="NOVAL"/>
<MeasZoneVals ID="4" ZoneType="SITE_NO" Val="0"/>
<MeasZoneVals ID="5" ZoneType="SITE_NO" Val="1"/>
<MeasZoneVals ID="6" ZoneType="SITE_NO" Val="2"/>
<MeasAllowedZoneTypes MeasAllowedID="1" ZoneType="NO_ZONE"/>
<MeasAllowedZoneTypes MeasAllowedID="1" ZoneType="SITE_NO"/>
<PosTypes ID="36" Name="CHIP" IsNormed="N"/>
<Testprgs ID="37" MeasAllowedID="1" Name="SSSSSSSSSSSSSSS" Rev="3.12" TdfStatus="missing"/>
<VirtualProductKeys ID="38" DWHProductCol="PRODUCT_TYPE"
DWHProductColVal="M1481N02314ZHOC"/>
<TUVirtualProductKey TUID="2" VirtualProductKeyID="38" DWHProductCol="PRODUCT_TYPE"/>
<TestprgVirtualProductKey TestprgID="37" VirtualProductKeyID="38"/>
<Measurements ID="39" MeasAllowedID="1" MeasCounter="1" CompSeqNr="0" TimestampBegin="2005-
05-10T22:51:13" TimestampEnd="2005-05-10T23:13:18" TimestampInsertion="2008-03-07T11:10:40"
TimestampLoading="2008-03-07T11:10:40" MeasCategory="PRODUCTIVE" TUID="2"
MeasLotID="02HN0P78" SubMeasStep="1" IsMaxSubMeasStep="Y" IsMaxMeasCounter="Y"
SumMethodName="ORIGINAL" TestType="N" ChipDataAttached="not_loaded"/>
<MeasTestprg MeasID="39" TestprgID="37" SeqNr="0"/>
<ParmDefs ID="40" MeasAllowedID="1" ParmType="HBIN" TestprgID="37" ParmNr="1"/>
</EXF>
```

Figure 24: Excerpt of a XML based EXF file generated by the STDF2EXF Converter

As one can see in the figure above, the EXF file is comprised by different records, each of them with a record type, such as `MeasAllowed` or `Testprgs`, and some related to other records through the use of foreign keys, as seen, for example, in the figure's blue line, where a `ParmDefs` record is connected to a `MeasAllowed` record through a `MeasAllowedID` and to a `Testprgs` record through a `TestprgID`.

Integration with EBS DB

The STDF2EXF Converter was successfully integrated with EBS DB, although a few implementation points were left out due to time constraints. All the logic from the STDF Loader was successfully migrated to the STDF2EXF Converter and the EXF Loader, except for the following requirements, regarding the loading of work packages into the staging area:

- *EBSDL_STDFLoader_151_EBS1096* – generation of automatic wafer numbers;
- *ITMFG00009511* – prevention of Oracle deadlocks when a `parameter_type` of a `parameter_definitions` record is updated in two parallel streams;
- *ITMFG00012868* – chip data attached update on the MEASUREMENTS table.

Such issues were migrated to the STDF2EXF Converter but the existence of conflicts that were not solved on the time allocated for the project left them as an open topic to be dealt with in the future.

The impact of the first issue on the system is considerably low, meaning that it was implemented in the STDF Loader with the purpose of handling situations that occur with low frequency. It concerns the generation of an automatic wafer number for work packages with "WAF" tracking unit type that have their wafer identifier as a non-numeric string. The amount of work packages that are included in this scenario are, however, as mentioned above, low.

The second and third issues have low/medium impact on the system, affecting a considerable amount of STDF work packages. They implicate the modification and creation of specific STDF related records, which affect the loading result of the majority of STDF work packages, although with not more than slight changes in comparison to the expected results. More information on this topic is available at 6.2 Testing phase 2 – integration with EXF Loader (regression testing).

With the situations described above controlled, meaning that although not implemented they do not affect the system's behaviour, EBS Data Loading can in this way now support the full loading of STDF work packages into the system through the new STDF2EXF Converter and the EXF Loader, a process which becomes now standardized for all raw data types.

Figure 25 shows an example of a database query through which it is possible to check the states of a set of work packages loaded into the system.

	WP_TR_ID	WP_STAT	WP_PHASE	WP_START	WP_STOP	WP_ACTJOB	WP_PATH
1	6	OK-ERROR	Done	08.03.03	08.03.03	JM-Emu	data/converters/TDF2EXF/error/T_TDW_XML
2	1	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/DT_N_TDF
3	2	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/TDF_Additional_Analysis
4	3	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/T_BTC_TDF
5	4	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/T_BTR_TDF
6	5	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/T_PCM_TDF
7	7	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/TDF_PCRBnumber
8	8	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/INV_NUMBER_TDF
9	9	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/WT_W5_TPR4_TDF2
10	10	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/WT_W5_TPR4_TDF1
11	11	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/WT_L_PCM_TDF
12	12	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/WT_L1_TCSV5_TDF
13	13	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/WT_ANYC_TDF
14	14	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/TDPW_TDF
15	15	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/TDF_TESTPROGRAMS
16	16	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/WT_WN_TDF
17	17	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/DT_N_D
18	18	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/DT_C_D
19	19	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/DT_C2_D
20	20	OK	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/save/DT_R_D
21	21	OK-ERROR	Done	08.03.03	08.03.03	JM-Emu	data/loader/EXF/error/DT_TR_D

Figure 25: States of work packages in EBS DB

Each work package, as seen above, is represented by a transaction id that uniquely identifies the work package being processed by the system. One can see if the work packages are loaded successfully – OK status – or with error – ERROR status – as well if the work packages were processed entirely or not, the component which is processing, among other useful information for an EBS DB user or administrator.

5.2 Detailed design

This section describes with some detail the prototype's structure and the sequence of actions that define its functioning.

Figure 26 shows the EBS DB file structure and, in particular, the files that compose the STDF2EXF Converter, which are described below.

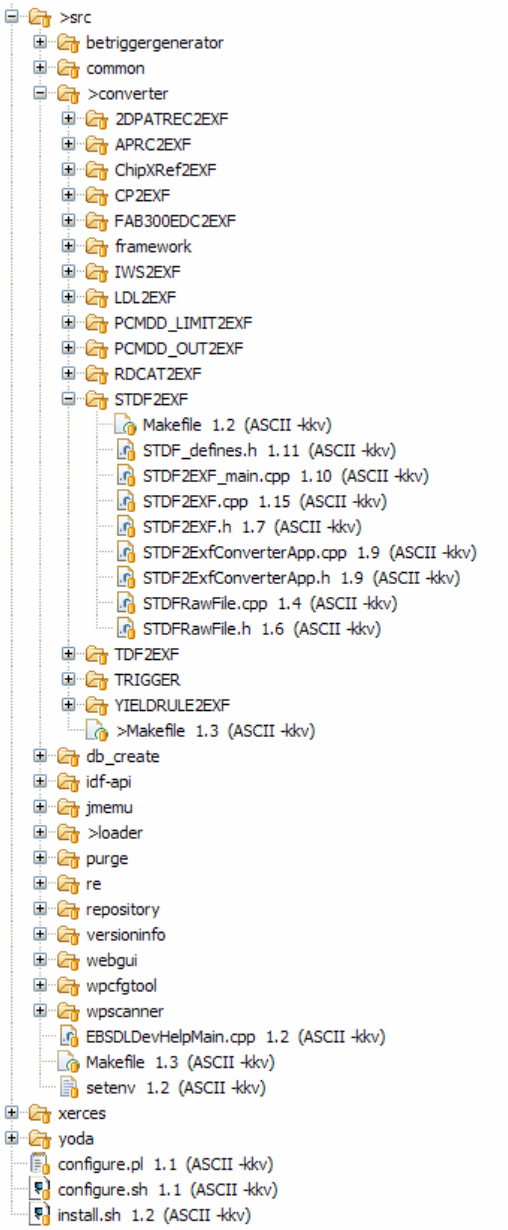

Figure 26: EBS DB file structure

STDF2EXF_main.cpp

This is the main class of the STDF2EXF Converter, initialized whenever a user or the JM Emulator calls the application. This can be done in two distinct ways, through the HP-UX command line:

- `STDF2EXF`
- `STDF2EXF -s <work package path>`

The former command is used by the JM Emulator when the converter is running integrated in EBS DB, whereas the latter is used by a user in the command line with the purpose of running the application in standalone mode, apart from an EBS DB installation. When called in this mode, the user shall provide the parameter `-s` followed by the path of a STDF work package that contains a raw file to be converted into EXF.

More commands can be used in the command line as follows:

- `STDF2EXF -v`
- `STDF2EXF --help`

The former shows the converter's compiled version while the latter provides a list of the different commands supported by the application.

When the application is called by the JM Emulator, it starts by initializing and setting up the Logging API – the EBS API that handles all its logging activities – together with the database access. Once this is achieved, this class launches a `CSTDF2EXFProcess` process that runs cyclically until the JM Emulator decides to terminate it. This process runs in a loop due to the fact that the converter can process several queued work packages sequentially or in parallel.

When called in standalone mode, the application executes some of the procedures mentioned above differently or skips them. Namely, the database access is not initialized since the application in this mode is autonomous and does not require database access, and the loggers are initialized in a different way, so that they don't require such database access. The converter process is launched with a standalone flag and is run once, instead of being called in a loop. This difference can be explained by the fact that when running in this mode, the converter processes only one work package in one run, opposed to the processing of multiple work packages when integrated with EBS DB.

STDF2EXF.cpp

The class `CSTDF2EXFProcess` represents the STDF2EXF Converter's operating system process called by the main class each time the converter runs. This process can run in a loop – when integrated with EBS DB – or run once – when in standalone mode. Whenever the converter is active, an instance of `CSTDF2EXFProcess` is running in the operating system.

This class is a subclass of a common EBS class denominated `CEbsProcess` which represents each process running in EBS. `CSTDF2EXFProcess` inherits from `CEbsProcess` the following methods:

- *runProcess()* – this function is commonly called by the main classes after instantiating a new process object. It initializes the process (see *initProcess()* below) and waits for incoming YODA messages from the JM Emulator, in particular one that includes a job request for the processing of a work package. When such event occurs, *processWP(int)* is called. The process is active until the JM Emulator sends it a termination instruction through a YODA shutdown message.
- *initProcess()* – initializes variables and the necessary APIs for the functioning of the converter – Work Package Management API, Configuration Management API and Logging API – as it loads the converter's configuration values from the repository database.
- *processWP(int)* – receives a work package id as argument and retrieves its information and configuration values to be used in the converting process from the repository database. Each work package scanned by EBS DB is listed in the repository database and the converter retrieves its information by accessing the appropriate table. It verifies if the work package is valid and creates input and output objects, prior to the launching of the conversion method.

- *releaseProcess()* – deletes variables and stops logging.

Such classes are standard in every converter and although they are inherited from `CEbsProcess`, their logic is defined in `CSTDF2EXFProcess` for all the previous methods except for *runProcess().*

This new converter has, however, a different logic when working in standalone mode and, in this way, new classes were created specifically for this converter, as follows:

- *runOnce(CEbsString)* – called by the main class after a new process object is instantiated in standalone mode. It does no more than calling the methods *initProcessStandalone(), processWPStandalone(CEbsString)* and *releaseProcess()* consecutively.
- *initProcessStandalone()* – initializes variables and loggers only, since API initialization is made using database access and this procedure is not necessary for the conversion of a single work package. It loads the converter's configuration values from .ini files stored locally.
- *processWPStandalone(CEbsString)* - verifies if the work package to process is valid and creates input and output objects, prior to the launching of the conversion function. The concept of work package id is not valid in standalone mode, since the work package to be processed is passed as argument in the command line and not listed in the repository database as it happens in the EBS DB mode.

STDF2EXFConverterApp.cpp

`STDF2EXFConverterApp` is responsible for handling the conversion process and the operations that precede and follow it. It inherits its methods from the converter framework class `CConverterApp` that is used by every converter.

It supports the creation and handling of input and output file objects, which represent respectively the raw STDF file to be converted and the destination EXF file. The conversion operations will be executed based in the input file object, while its results are temporarily stored in the output file object and afterwards saved in the EXF file.

The following methods are part of this class and worth being mentioned:

- *createInputFileObject(…)* – creates an object for the raw data input file and sets its variables with the configurations previously retrieved from the repository database.
- *createOutputFileObject(…)* - creates an object for the EXF output file.
- *convert(long)* – calls *processFile()* from `CSTDFRawFile`, *cbPreConvert()* and *cbPostConvert()* methods before and after the previous call, respectively.
- *cbPreConvert()* – opens the EXF file object for writing before conversion occurs.
- *cbPostConvert()* – closes the EXF file object after conversion and writing.

STDFRawFile.cpp

`CSTDFRawFile` contains all the logic regarding the transformation process of the STDF raw file into a XML based EXF file.

The same class in the STDF Loader contains logic regarding not only the conversion process of raw data into EXF structured data, but also the loading of such information into the staging database. In order to migrate the conversion logic into the STDF2EXF Converter, there was a need to select and arrange the existing code in a way that only the conversion logic would be incorporated into the converter.

6 Project concretisation

6.1 Testing phase 1 – standalone mode

Once the first Design Engineering Phase was completed and the STDF2EXF Converter prototype designed to be used in standalone mode, a testing phase followed with the goal of validating the work that had been done, assuring that the prototype had been designed according to its requirements. The result of such tests was a document entitled System Test Report [20], which was created and filled according to Critical Software's Quality guidelines.

All tests were based in the test cases previously defined in the Test Case Specification [19] and performed in the HP-UX operating system environment. A part of the tests involved comparing results between the new STDF2EXF Converter and the existing STDF Loader. The STDF Loader version used in the test was part of version 2.3.0.3 of EBS DB.

A test was considered as successful if the application behaviour was consistent with the test case's specified output. If the application did not behave as specified, the test was considered as failed.

The following scale was the base for the classification of the identified problems (faults), as defined in [19]:

- Level 1. Serious fault that compromised the use of the system or a relevant part of the system.
- Level 2. Fault which implied that functionality was not fully achieved but did not prevent the system from being used.
- Level 3. Fault that did not compromise the use of the system and did not represent a loss in functionality. Typically these faults are problems with the GUI (Graphical User Interface).

The specified test cases were carried on in three test runs. Some changes were performed on the source code that needed test case `STDF2EXF-TCS-LOG-001` to be retested, after Test Run 1 was finished. This was performed in Test Run 2. Test Run 3 was executed due to a change on the test case `STDF2EXF-TCS-VAL-004 XML data integrity`.

In this way, 16 tests were executed, all of them ending with success, meaning that the STDF2EXF Converter was tested successfully and is ready to be used in its standalone mode.

Below are shown the results of the executed tests. Each test run was performed over a CVS testing tag, so that test results could be controlled effectively and changes tracked successfully between test runs. This is a standard procedure in different kinds of software projects inside Critical Software.

Test run 1

The first test run was executed as detailed in Table 3 and its results are shown below in Table 4.

Test Case Execution	
Run date:	2007-12-20
User:	Daniel Silva
CVS tag (build):	EBSDBLGC-BUI-STDF2EXF-0_1-TEST-06
Test specification reference	CSW-EBSDBLGC-2007-TCS-7636

Table 3: Test run 1 data

Test ID	Success (S)/ Failure (F)	Error level (1/2/3)	Problems found
STDF2EXF-TCS-WRK-001	S		
STDF2EXF-TCS-VAL-007	S		
STDF2EXF-TCS-VAL-008	S		
STDF2EXF-TCS-VAL-005	S		
STDF2EXF-TCS-VAL-006	S		
STDF2EXF-TCS-VAL-001	S		
STDF2EXF-TCS-VAL-002	S		
STDF2EXF-TCS-LOG-001	S		
STDF2EXF-TCS-CON-001	S		
STDF2EXF-TCS-CFG-001	S		
STDF2EXF-TCS-CFG-002	S		
STDF2EXF-TCS-STR-001	S		
STDF2EXF-TCS-VAL-004	S		
STDF2EXF-TCS-VAL-003	S		

Table 4: Test run 1 results

Test run 2

The second test run was executed as detailed in Table 5 and its results are shown below in Table 6.

Test Case Execution	
Run date:	2007-12-20
User:	Daniel Silva
CVS tag (build):	EBSDBLGC-BUI-STDF2EXF-0_1-TEST-07
Test specification reference	CSW-EBSDBLGC-2007-TCS-7636

Table 5: Test run 2 data

Test ID	Success (S)/ Failure (F)	Error level (1/2/3)	Problems found
STDF2EXF-TCS-LOG-001	S		

Table 6: Test run 2 results

Test run 3

The third and last test run was executed as detailed in Table 7 and its results are shown below in Table 8.

Test Case Execution	
Run date:	2008-01-03
User:	Daniel Silva
CVS tag (build):	EBSDBLGC-BUI-STDF2EXF-0_1-TEST-07
Test specification reference	CSW-EBSDBLGC-2007-TCS-7636

Table 7: Test run 3 data

Test ID	Success (S)/ Failure (F)	Error level (1/2/3)	Problems found
STDF2EXF-TCS-VAL-004	S		

Table 8: Test run 3 results

6.2 Testing phase 2 – integration with EXF Loader (regression testing)

The regression tests are performed to verify the correct work of the data loading components of EBS DB. They are executed before each release and consist in loading several work packages into the system to assure the it behaves as expected.

A test framework, known as regression framework, was created in Critical Software for this purpose, allowing one to run a series of tests regarding the loading of specific work packages into the system. One can define the work packages to be loaded, such as memory (master data) or fact data work packages.

The results are obtained by performing a database extraction of the data loaded through the work packages and comparing it to the expected results generated in the previous version. Each extraction portraits the information that was loaded into the system per work package, meaning that each fact data work package will have its extract, which can then be compared to the same extract performed in the previous version. If no significant changes were made in the new version, then the generated extraction files of both versions shall be identical. If differences occur, they should be justified.

The regression test framework comprises several test cases, each consisting on loading a work package into EBS DB. Table 9 contains the list of test cases executed in EBS DB modified version 2.3.0.3 that includes the STDF2EXF Converter.

Test Case Name	WP Name	Converter
TDF000	DT_N_TDF	TDF
TDF001	INV_NUMBER_TDF	TDF
TDF002	IT_BTC_TDF	TDF
TDF003	IT_BTR_TDF	TDF
TDF004	IT_PCM_TDF	TDF
TDF005	IT_TDW_XML	TDF
TDF006	TDF_Additional_Analysis	TDF
TDF007	TDF_PCRBnumber	TDF
TDF008	TDF_TESTPROGRAMS	TDF
TDF009	TDPW_TDF	TDF
TDF010	WT_ANYC_TDF	TDF
TDF011	WT_L1_TCSV5_TDF	TDF
TDF012	WT_L_PCM_TDF	TDF
TDF013	WT_W5_TPR4_TDF1	TDF
TDF014	WT_W5_TPR4_TDF2	TDF
TDF015	WT_WN_TDF	TDF
STDF000	DT_C2_D	STDF
STDF001	DT_C_D	STDF
STDF002	DT_N_D	STDF
STDF003	DT_R_D	STDF
STDF004	DT_TR_D	STDF
STDF005	INV_NUMBER_D	STDF
STDF006	IT_BTA_D	STDF
STDF007	IT_BTC_D	STDF
STDF008	IT_BTP_D	STDF
STDF009	IT_BTR_D	STDF
STDF010	IT_PCM_TR	STDF
STDF011	IT_PCM_WAF_1_D	STDF
STDF012	IT_PCM_WAF_2_D	STDF
STDF013	IT_SMS_NSMR_D	STDF
STDF014	STDF_Additional_Analysis	STDF
STDF015	STDF_PCRBnumber	STDF
STDF016	STDF_TEST_WP_01	STDF
STDF017	STDF_TEST_WP_08	STDF
STDF018	STDF_TEST_WP_09	STDF
STDF019	STDF_TEST_WP_33	STDF
STDF020	TDPW_WDNE_D	STDF
STDF021	WT_ANYC_N_D	STDF
STDF022	WT_L1_TCSV5_D1	STDF
STDF023	WT_L1_TCSV5_D2	STDF
STDF024	WT_L1_TCSV5_D3	STDF
STDF025	WT_L5_TPR_TR1	STDF
STDF026	WT_L5_TPR_TR2	STDF
STDF027	WT_L_ANYC_N_D	STDF
STDF028	WT_L_ANYC_TR	STDF
STDF029	WT_L_PCM_D	STDF
STDF030	WT_L_PCM_TR	STDF
STDF031	WT_L_TR	STDF
STDF032	WT_TR_1	STDF
STDF033	WT_TR_3	STDF
STDF034	WT_TR_6	STDF
STDF035	WT_TR_9	STDF

STDF036	WT_TR_SMS-1_MC-P_W-1_D	STDF
STDF037	WT_TR_SMS-1_MC-P_W-4_D	STDF
STDF038	WT_TR_SMS-1_MC-P_W-5_D	STDF
STDF039	WT_TR_SMS-1_MC-S_W-1_D	STDF
STDF040	WT_TR_SMS-1_MC-S_W-4_D	STDF
STDF041	WT_TR_SMS-1_MC-S_W-5_D	STDF
STDF042	WT_TR_SMS-1_MC-X_W-1_D	STDF
STDF043	WT_TR_SMS-1_MC-X_W-3_D	STDF
STDF044	WT_TR_SMS-1_MC-X_W-4_D	STDF
STDF045	WT_TR_SMS-2_MC-P_W-2_D	STDF
STDF046	WT_TR_SMS-2_MC-P_W-4_D	STDF
STDF047	WT_TR_SMS-2_MC-P_W-5_D	STDF
STDF048	WT_TR_SMS-3_MC-P_W-2_D	STDF
STDF049	WT_TR_SMS-3_MC-P_W-3_D	STDF
STDF050	WT_TR_SMS-3_MC-P_W-5_D	STDF
STDF051	WT_TR_SMS-3_MC-S_W-1_D	STDF
STDF052	WT_TR_SMS-3_MC-S_W-2_D	STDF
STDF053	WT_TR_SMS-3_MC-S_W-5_D	STDF
STDF054	WT_W5_TPR4_D1	STDF
STDF055	WT_W5_TPR4_D2	STDF
STDF056	WT_WC_D	STDF
STDF057	WT_WN_D	STDF
STDF058	WT_WR_D	STDF
STDF059	WT_WTR_D	STDF
STDF060	S_20061114055255_PF702488_S11P_N.st4_005_VIH	STDF

Table 9: Test cases for regression testing

The regression testing was performed using master data TDF work packages and fact data STDF work packages. Some work packages had to be loaded in a specific order so that their data would be loaded correctly, assuring linked data was preserved.

Test run 1

The first and only test run was executed as detailed in Table 10 and its results are shown below.

Test Case Execution	
Run date:	2008-02-28
User:	Daniel Silva
CVS tag (build):	EBSDBLGC-BUI-STDF2EXF-0_1-TEST-09
Test specification reference	CSW-EBSDBLGC-2008-TCS-01687

Table 10: Regression test run 1 data

All tests were successful, although justifiable differences were detected. Such differences were present constantly throughout the different database extracts, and related to the non-implementation of previously existing functionalities in EBS DB. Such differences will be exemplified below regarding a single work package and are valid for the remaining work packages loaded in the regression tests, as their extracts have similar differences, resulting from the same identified causes.

updateChipDataAttached()

The first two differences are noticed for all work packages that contain attached chip data information. One of the three functionalities that were not implemented in the STDF2EXF Converter and the EXF Loader concerns the handling of chip data.

In EBS DB v2.0, a column was created in the MEASUREMENTS table in the EBS_ANA database with the aim of being filled with ‘`Y`’ whenever chip data has been loaded for a measurement or ‘`not_loaded`’ if the loader has not loaded this data into the database. The function *updateChipDataAttached()*, implemented in STDF Loader’s `CSTDFRawFile` class, was not successfully migrated to the STDF2EXF Converter and thus not fully implemented in this modified version of EBS DB.

Figure 27 shows the differences that were found.

Report_WP_DT_C2_D.lst – Original v2.3.0.3 file

```
Measurements table
TYPE ;MEASSTP_NAME ;MEASDS_NAME ;SUB_MEAS_STEP ;MEAS_LOT_ID ;MEASCAT_NAME
;MEAS_COUNTER;TEST_TYPE ;COMPONENT_SEQ;TEST_TYPE ;T_BE_INS ;T_LOAD ;CHIP_ATTAC
LOT ;B2 ;FCT BE ;1 ;VE527626M12 ;PRODUCTIVE ; 1;C ; 1;C ; ;28-FEB-08;Y
```

Report_WP_DT_C2_D.lst – Modified v2.3.0.3 file

```
Measurements table
TYPE ;MEASSTP_NAME ;MEASDS_NAME ;SUB_MEAS_STEP ;MEAS_LOT_ID ;MEASCAT_NAME
;MEAS_COUNTER;TEST_TYPE ;COMPONENT_SEQ;TEST_TYPE ;T_BE_INS ;T_LOAD ;CHIP_ATTAC
LOT ;B2 ;FCT BE ;1 ;VE527626M12 ;PRODUCTIVE ; 1;C ; 1;C ; ;28-FEB-08;not_loaded
```

Figure 27: Database update through *updateChipDataAttached()* – `CHIP_ATTAC` value

The original database extract, *Report_WP_DT_C2_D.lst*, regarding the STDF work package *DT_C2_D*, indicates that chip data was loaded, while the new generated report for the same work package has such value equal to ‘`not_loaded`’ due to the fact that such data is not processed by the STDF2EXF Converter as this functionality was not implemented.

The fact that this function was not implemented will cause the extracts to show additional differences, as shown in Figure 28 and Figure 29.

Report_WP_DT_C2_D.lst – Original v2.3.0.3 file

```
Measpart_Attribute table
TYPE ;MEASSTP_NAME ;MEASDS_NAME ;SUB_MEAS_STEP ;MEAS_LOT_ID ;PARTATTRTP_NAME ;VALUE_T
LOT ;B2 ;FCT BE ;1 ;VE527626M12 ;IS_LAST_VALID ;VARCHAR2_32_V('Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y')
LOT ;B2 ;FCT BE ;1 ;VE527626M12 ;test-count ;VARCHAR2_32_V('1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1')
```

Report_WP_DT_C2_D.lst – Modified v2.3.0.3 file

```
Measpart_Attribute table
TYPE ;MEASSTP_NAME ;MEASDS_NAME ;SUB_MEAS_STEP ;MEAS_LOT_ID ;PARTATTRTP_NAME ;VALUE_T
```

Figure 28: Database update through *updateChipDataAttached()* – Measpart Attributes

When attached chip data is not updated, some values are not loaded into the Measpart Attributes table, such as the `IS_LAST_VALID` and `test-count` attributes.

Report_WP_DT_C2_D.lst – Original v2.3.0.3 file

```
Environment_Values table
TYPE ;MOTHERLOT_ID ;MEAS_LOT_ID ;SUB_MEAS_STEP ;MEASSTP_NAME ;MEASDS_NAME ;
SEQ_NR;ENVTP_NAME ;ZONETP_NAME ;ZONE_VALUE ;VALUE
LOT ;VE527626 ;VE527626M12 ;1 ;B2 ;FCT BE ; 0;Component_Merge_Info ;NO_ZONE ;NOVAL ;merged
into lot aggregation
```

Report_WP_DT_C2_D.lst – Modified v2.3.0.3 file

```
Environment_Values table
TYPE ;MOTHERLOT_ID ;MEAS_LOT_ID ;SUB_MEAS_STEP ;MEASSTP_NAME ;MEASDS_NAME ;
SEQ_NR;ENVTP_NAME ;ZONETP_NAME ;ZONE_VALUE ;VALUE
```

Figure 29: Database update through *updateChipDataAttached()* – Enviroment Values

The picture above shows that some environment values might not be loaded, in particular `Component_Merge_Info`, as a result of such data not being updated.

The assumptions made above were based on parallel regression testing made on the original v2.3.0.3 build by disabling the call to the *updateChipDataAttached()* function in the STDF Loader's code. The extractions of this version and the one with the STDF2EXF Converter were identical.

updateAllParameterTypes()

Occasionally, the parameter type of parameter definitions has to be updated, changing between ANA (analytical) and FCT (functional). This is done in v2.3.0.3 by calling the function *updateAllParameterTypes()* from the STDF Loader's `CSTDFRawFile` class.

In some work packages, a minor percentage of records – less than 5% – might have their parameter type set incorrectly, since this functionality was not yet implemented into the STDF2EXF Converter and some parameter types are not updated as it happens in v2.3.0.3.

Differences as seen on Figure 30 might affect not only Parameter Aggregates records but also, in some cases, Measpart Values records.

Report_WP_DT_C2_D.lst – Original v2.3.0.3 file

```
Parameter_Aggregates table
TYPE ;MEASSTP_NAME ;MEASDS_NAME ;SUB_MEAS_STEP ;MEAS_LOT_ID ;MEASCAT_NAME
;MEAS_COUNTER;TEST_TYPE ;COMPONENT_SEQ;PARTP_NAME ;NAME ; PNUMBER;ZONETP_NAME ;VALUE ; EXEC;
MIN; MAX; MEAN; MEDIAN; STDEV; Q01; Q02; Q05; Q10; Q15; Q25; Q75; Q85; Q90; Q95; Q98; Q99;
IQR; RANGE; FAIL; PASS;RDCAT_AGGR ;INLINE_AGGR;SPEC_YIELD;VALID_YIELD;
ALL_YIELD;BASE_CHIPS;VALID_CHIPS;SPEC_CHIPS;BASE_TYPE ; COUNT; SUM;
SQR_SUM;FAIL_GEO_RATIO;FAIL_TOTAL_RATIO;EXEC_GEO_RATIO;EXEC_TOTAL_RATIO;FAIL_EXEC_RATIO;COUN
T_GEO_RATIO;COUNT_TOTAL_RATIO
LOT ;B2 ;FCT BE ;1 ;VE527626M12 ;PRODUCTIVE ; 1;C ; 1;ANA ;XKT1 ; 1;NO_ZONE ;NOVAL ; ; 0; 8;
0; 0; 0; ; ; 0; 0; ; ; ; ; 0; 0; ; ; ; ; 1; ; ; ; ; ; ; 203; ; ;P ; ; 8; 64; ; .054229935; ;
; ; ;
```

Report_WP_DT_C2_D.lst – Modified v2.3.0.3 file

```
Parameter_Aggregates table
TYPE ;MEASSTP_NAME ;MEASDS_NAME ;SUB_MEAS_STEP ;MEAS_LOT_ID ;MEASCAT_NAME
;MEAS_COUNTER;TEST_TYPE ;COMPONENT_SEQ;PARTP_NAME ;NAME ; PNUMBER;ZONETP_NAME ;VALUE ; EXEC;
MIN; MAX; MEAN; MEDIAN; STDEV; Q01; Q02; Q05; Q10; Q15; Q25; Q75; Q85; Q90; Q95; Q98; Q99;
IQR; RANGE; FAIL; PASS;RDCAT_AGGR ;INLINE_AGGR;SPEC_YIELD;VALID_YIELD;
ALL_YIELD;BASE_CHIPS;VALID_CHIPS;SPEC_CHIPS;BASE_TYPE ; COUNT; SUM;
SQR_SUM;FAIL_GEO_RATIO;FAIL_TOTAL_RATIO;EXEC_GEO_RATIO;EXEC_TOTAL_RATIO;FAIL_EXEC_RATIO;COUN
T_GEO_RATIO;COUNT_TOTAL_RATIO
LOT ;B2 ;FCT BE ;1 ;VE527626M12 ;PRODUCTIVE ; 1;C ; 1;FCT ;XKT1 ; 1;NO_ZONE ;NOVAL ; ; 0; 8;
; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; 1; ; ; ; ; ; ; ; ;A ; ; 8; 64; ; .054229935; ; ; ; ;
```

Figure 30: Database update through *updateAllParameterTypes()* – Parameter Aggregates

6.3 Results

Throughout the project, there was a concern in documenting all the work in conformity with Critical Software's quality standards. Various documents were created, being the first one a Software Requirements Specification [11] that included the description of functional and non-functional requirements the solution should be based on, its use cases catalogue, a domain model and sequence and state diagrams. A Test Case Specification [19] was produced afterwards, including the test cases catalogue and traceability matrixes connecting such test cases with requirements and use cases. This document was the basis for all the software and system testing.

A functional vertical prototype was designed and tested, based, respectively on the gathered requirements and the specified test cases. This prototype incorporated some of the STDF Loader's logic and was built based on an existing converter template. The prototype was tested in two distinct phases: the first one had the goal of testing the STDF2EXF Converter's standalone mode, while the second one tested its working mode integrated in EBS.

The first test phase consisted in checking that the converter executed its conversion mechanisms accordingly to what had been specified, without errors, and that the application working modes had been implemented as expected. The behaviour of the application was constant, having all tests finished with success. This first phase involved as well converting a set of predefined STDF work packages and comparing the results with the expected outputs obtained with the EBS DB v2.3.0.3 release. All conversions were executed successfully and the new generated output files were identical to the expected ones. These tests indicated that the converter is fully prepared to be used in standalone mode.

The second test phase involved testing the integration of the STDF2EXF Converter with the EXF Loader and EBS DB, by running the EBS DB regression tests, which allowed one to verify the information loaded into the EBS databases by introducing work packages into the new workflow comprising the STDF2EXF Converter. The tests involved extracting the information from the EBS databases and comparing such extractions with the ones previously executed in the v2.3.0.3 release. The results matched the expectations, having in mind that some of the loading functionalities of the STDF Loader were not migrated into the STDF2EXF Converter and the EXF Loader.

These tests showed, in an EBS DB system processing STDF work packages with the STDF2EXF Converter and the EXF Loader (contrary to the existing procedure with the STDF Loader), the following results:

- 100% of the STDF work packages were converted as expected, with the converted information being 100% accurate;
- 100% of the STDF work packages were loaded as expected, with the loaded information being between 99% and 99.9% accurate in 78% of the cases, between 97% and 98.9% accurate in 16% of the cases, and below 97% accurate in 6% of the cases.

A Regression Testing Specification [21] document was produced, explaining how such tests were performed and their main goals.

The system behaved and performed as expected, being the measured conversion and loading times similar to the ones observed in version 2.3.0.3.

6.4 Project review

This section summarizes the real work that was done by describing a Gantt chart showing the performed tasks, their duration, their start and finish times.

ID	WBS	Task Name	Duration	Start	Finish
1	DS	**Daniel Internship PLN**	**22 wks**	**Mon 01-10-07**	**Fri 07-03-08**
2	WP-1000	**Business Know-How**	**0,8 wks**	**Mon 01-10-07**	**Thu 04-10-07**
3	WP-1100	EBS Data Loading	1 day	Mon 01-10-07	Mon 01-10-07
4	WP-1200	EBS Data Model	0,5 days	Tue 02-10-07	Tue 02-10-07
5	WP-1300	Converters Framework	0,5 days	Tue 02-10-07	Tue 02-10-07
6	WP-1400	STDF Format	0,5 days	Wed 03-10-07	Wed 03-10-07
7	WP-1500	EXFLoader	0,5 days	Wed 03-10-07	Wed 03-10-07
8	WP-1600	STDFLoader	1 day	Thu 04-10-07	Thu 04-10-07
9	WP-2000	**Requirements Engineering Phase**	**2 wks**	**Mon 08-10-07**	**Fri 19-10-07**
10	WP-2100	Software Requirement Specification	5 days	Mon 08-10-07	Fri 12-10-07
11	WP-2200	Test Case Specification	5 days	Mon 15-10-07	Fri 19-10-07
12	WP-3000	**Phase 1**	**10,8 wks**	**Mon 22-10-07**	**Tue 08-01-08**
13	WP-3100	**Design Engineering Phase**	**7,6 wks**	**Mon 22-10-07**	**Thu 13-12-07**
14	WP-3110	Code implementation	38 days	Mon 22-10-07	Thu 13-12-07
15	WP-3200	**Validation Phase**	**1,2 wks**	**Fri 14-12-07**	**Fri 21-12-07**
16	WP-3210	System Testing	3 days	Fri 14-12-07	Tue 18-12-07
17	WP-3220	Update documentation	3 days	Wed 19-12-07	Fri 21-12-07
18	WP-3300	**Internship Documentation**	**2 wks**	**Mon 24-12-07**	**Tue 08-01-08**
19	WP-3310	Write documentation	10 days	Mon 24-12-07	Tue 08-01-08
20	WP-4000	**Phase 2**	**8,4 wks**	**Wed 09-01-08**	**Fri 07-03-08**
21	WP-4100	**Design Engineering Phase**	**4,8 wks**	**Wed 09-01-08**	**Tue 12-02-08**
22	WP-4110	Code implementation	24 days	Wed 09-01-08	Tue 12-02-08
23	WP-4200	**Validation Phase**	**0,4 wks**	**Wed 13-02-08**	**Thu 14-02-08**
24	WP-4220	Update documentation	2 days	Wed 13-02-08	Thu 14-02-08
25	WP-3300	**Internship Documentation**	**3,2 wks**	**Fri 15-02-08**	**Fri 07-03-08**
26	WP-3310	Write documentation	12 days	Fri 15-02-08	Fri 07-03-08
27	WP-5000	**Performance Study**	**0,8 wks**	**Wed 27-02-08**	**Mon 03-03-08**
28	WP-5100	Run Regression Tests	3 days	Wed 27-02-08	Fri 29-02-08
29	WP-5200	Compare Results	1 day	Mon 03-03-08	Mon 03-03-08

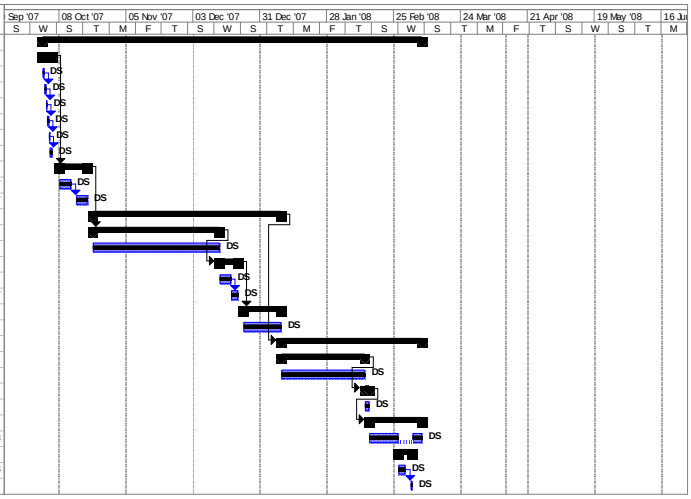

Figure 31: Gantt chart with the real executed tasks

The project was divided into nine main tasks, included in five distinct phases.

The initial phase involved getting familiar with the EBS system, namely its purpose, its structure and the semiconductor manufacturing process. The business know-how was gained through formation sessions that were carried out with slide presentations and some code insights, together with individual study complementary to such sessions. The EBS DB was in this way introduced through a series of steps, being the modules with the most interest for the project presented in appropriate detail. The different tasks seen on the plan for this phase were executed in sequence, although not following the indicated order at all times.

The Requirements Engineering Phase was initialized one week after the kick-off meeting, with the writing of the Software Requirements Specification [11] and the Test Case Specification [19] documents. This phase involved understanding in detail the problem and gathering the fundamental requirements for the system, in order to establish a solid base from where the problem solution could be progressively built. The requirements specification was based on discussion between the persons involved in the project – trainee, tutor and project

manager – following the main topics defined in the project's kick-off meeting. Once the requirements were defined and still prior to the initial coding, the system was modelled using UML use case, class, state and sequence diagrams. These models served as a starting point and high level sketches to the coding process that would follow during the next weeks. Not less important was the definition of test cases that would assure that the established requirements had been satisfied after coding completion.

The first coding phase comprised the STDF2EXF Converter initial prototype design, with the goal of having a fully functional version of this prototype working in standalone mode by the end of the phase. This was done by incorporating the conversion logic of the existing STDF Loader into a new STDF2EXF Converter structured according to an EBS DB converter template. The prototype was developed in less than eight weeks, being tested and validated in mid December. The project report writing process followed the prototype testing and validation but was interrupted in the beginning of January, after being decided that the prototype to be presented in the end of the project should have the capability of working integrated in EBS DB as well. This decision was based, as mentioned previously in this report, on changes to MIEIC's Project regulations.

The second coding phase comprised, thus, the integration of the STDF2EXF Converter with EBS DB, by migrating the loading logic of the STDF Loader into the EXF Loader. Contrary to Phase 1, the converter and the EXF Loader were not fully tested, since not all their functionalities were successfully implemented.

The Software Requirements Specification [11] and Test Case Specification [19] documents were updated after the development phases, with new requirements that were born throughout the project, changes to the previous models and catalogues, together with other relevant information.

The last weeks of the project comprised the report writing, the EBS regression testing and the deliverables handling.

Other projects

The time allocated for the author to carry out this project was not constant at all times, with the existence of periods where the focus on the project would drop to 50%, due to the integration of the trainee in Critical Software's EBS DB maintenance team and the EBS-DB-LOGIC project.

From December 2007 to March 2008, and in parallel to this project, the trainee was involved in three releases of EBS DB. The trainee was involved in version 2.4.1.0 (specification, implementation, testing), version 2.4.2.0 (specification, implementation, testing, release tasks) and in version 2.5.0 (specification). Such tasks consumed a period of over two weeks – approximately 12 days – throughout the project's time – between October 2007 and February 2008.

Plan comparison

Estimated to be carried out in 22 weeks, the plan was executed as expected and its major and high priority tasks were achieved successfully. The project lasted for 23 weeks and it included parallel activities not part of the project nor its plan, starting from December 2007 (as mentioned above in Other projects). By subtracting such parallel work – 12 days – to the total

project duration – 23 weeks – it can be concluded that the project's real work was done in less than 21 weeks, thus, fulfilling what had been defined in MIEIC's Project Regulations, where it is stated that the Project's duration shall be of 20 weeks.

Although there have been slight differences between what was planned and what was indeed executed, the plan was in general followed and the project was completed on time. Comparing what has been done with the plan, some conclusions can be drawn. The development phases took longer than expected – 38 days against the 30 days initially estimated for the first coding phase, 24 days against the initially estimated 15 days for the second coding phase – together with the project's report writing – 22 days against the initially estimated 10 days. On the contrary, Business Know-How and the Requirements Engineering Phase took about half of the expected time to be fulfilled – one week against the two weeks estimation and two weeks against the four weeks estimation, respectively – even though some of these tasks were at some points resumed throughout the project. The system testing of Phase 2 and the performance report writing were skipped as such tasks were not valuable due to the current system state, which does not have all its functionalities implemented at this time.

One shall notice that the durations indicated above do not necessarily indicate the real effort put into such tasks but the total time in days or weeks such tasks lasted until they were completed, e.g., a task which lasted 10 days might have been done with an effort of 60%, meaning 6 days were required to complete it, while 4 days were used in another project, not included in the plan but still affecting the time taken to complete the task.

7 Conclusions and future work perspective

The “STDF2EXF Converter Development” project at Critical Software was successful in various levels, with its main goals being reached and matching the initial stakeholders expectations. The project was completed on time even though the plan was changed significantly comparatively to what had been defined in the beginning of the project.

This was an experimental project that aimed to introduce into EBS DB a new way of processing information, through the removal of the existing STDF Loader component and the introduction of a converter that would process STDF binary files as other existing converters process their own text file types. By doing this the system would be comprised by a single loader and the processing of all raw files would be identical regardless of the binary or text nature of the input data. Besides making the system more homogeneous and possibly increasing its performance, the project had also the goal of developing a converter that could work in standalone mode, apart from EBS DB, added to the normal behaviour of converters that work integrated with EBS DB.

The implementation of the converter in standalone mode was straightforward and entirely successful. It was developed according to a Software Requirements Specification [11], a document produced after discussion among the project’s stakeholders about the main needs of the application and the way it should behave. This document included the main software requirements for the application together with useful diagrams modelled in UML.

Some problems arose during the second stage of the converter’s implementation, regarding its integration with the EXF Loader. Due to the existence of process conflicts when running the system with some of the STDF Loader’s functionalities, some methods were not migrated successfully until the development end time of the project. These functionalities, which could possibly be successfully migrated in a matter of a couple of weeks, have a low impact on the system and can not be seen as alarming regarding its functioning as it has been developed. Below are shown the items not migrated from the STDF Loader into the STDF2EXF Converter and/or EXF Loader in this project:

- *EBSDL_STDFLoader_151_EBS1096* – generation of automatic wafer numbers;
- *ITMFG00009511* – prevention of Oracle deadlocks when a `parameter_type` of a `parameter_definitions` record is updated in two parallel streams;
- *ITMFG00012868* – chip data attached update on the MEASUREMENTS table.

The prototype of EBS DB with the STDF2EXF Converter processed STDF files behaving and performing as expected, with no significant differences in the work packages conversion and loading times, even though this was executed in a preliminary build of the system without the functionalities described above.

The system was tested in two phases. In a first stage, after the initial development phase, the STDF2EXF Converter was tested as an isolate component, in its standalone mode, while in a later stage this component was integrated with the EBS DB system. The former aimed to test the converter’s STDF conversion process while the latter had the goal of testing the STDF loading process of EBS DB – regression testing. These tests were based, respectively, in a Test Case Specification [19] – based in the previous Software Requirements Specification [11] document – and in a Regression Testing Specification [21].

All performed tests returned the expected results, regarding the implemented functionality by the date tests were carried on.

The STDF2EXF Converter is ready to be used in standalone mode and a build has been released for internal usage. This release is used by the EBS DB team for testing purposes involving STDF work packages, allowing members to quickly handle and edit STDF files used in tests. This application brings such users a major advantage when compared to the previous existing solution for the matter, which involved manually editing the STDF Loader code and compile it specifically to perform this action. This used to be done by changing a flag in the STDF Loader, followed by a rollback when EXF generation would not be needed anymore so that the STDF Loader would behave as expected inside the EBS DB.

By having a version of the STDF2EXF Converter compiled and compressed in an extractable file, any user can download this application to a place of his choice and quickly convert any STDF work package in a matter of seconds, without having the need of installing an EBS build, compiling components, or launching work packages into the EBS DB workflow. The standalone mode makes the conversion process simple, fast and straight into the user expectations and aims, which is to access information in an easy, quick and reliable way. It is a tool that optimizes the process of accessing and modifying STDF data and, although not crucial for the success of a project, it is definitely a time saver and an indispensable tool to have daily at hand in such a development team in Critical Software.

Prototype completion

The next logic step to carry out within the project's scope would be to integrate the STDF loading functionalities that were not implemented in this build with the EXF Loader. This issue was aborted due to the existence of process conflicts when executing such operations after an approach to incorporate such logic into the STDF2EXF Converter, having not been solved in proper time.

Once such items are successfully incorporated into the prototype, the STDF2EXF Converter application will then be indeed fully integrated with EBS DB, in comparison to version 2.3.0.3 of this system, presenting together with the EXF Loader the same functionalities the STDF Loader does in this version.

EXF Loader testing

Although the regression testing is a reliable mechanism to verify the work packages loading process, specific testing shall be made regarding the EXF Loader's operation, similarly to what was executed for the STDF2EXF Converter.

The STDF Loader loading test cases had been previously defined by the EBS DB team as part of the system's regular testing and were in this project adapted – when applicable – to be used in the EXF Loader's testing for the STDF format. Instead of comparing the whole results of the work packages loading, such test cases allow a deeper analysis of the component itself and a different opportunity of detecting possible flaws. Each test case has the goal of checking if one or a set of functionalities are correctly implemented and this, added to the general picture provided by the regression tests, would complete the testing needed to be done regarding the loading of STDF work packages.

The EXF Loader testing shall be carried out once all the STDF Loader's functionalities have been migrated successfully.

Performance testing

One of the goals of the project, if time permitted, was to eventually compare how a system without the STDF Loader would behave compared to the solution that was adopted and is being currently used by Infineon Technologies. The transformation of a system with two loaders into one with a single loader that handles all file types would have an impact on the system due to the importance and complexity of such components in the process. Such impact might be positive or negative, although it is clear that the system is more homogeneous with the new solution, having a single EXF Loader instead of this component being complemented with a specific loader that handles STDF files. From a coding point of view, not much new code was produced, having happened instead a migration of the STDF Loader's code, its rearrangement and slight optimization when integrated into the STDF2EXF Converter and the EXF Loader.

Performance testing was not executed at this time, since the new developed system does not include all the loading logic of the STDF Loader. Once this is achieved, performance tests shall be carried out, results compared and a report shall be produced.

Update to current EBS DB version

Once completed the steps mentioned above, new conclusions can be drawn from the obtained results. If the proposed solution is indeed worthy to be exploited and adopted in the future by Critical Software and Infineon Technologies, all the changes since version 2.3.0.3 until the current EBS DB version (currently 2.4.2.0) will eventually need to be incorporated into the system that has been developed. In particular, all the logic added to the STDF Loader since version 2.3.0.3 shall be migrated to the STDF2EXF Converter and EXF Loader, depending on its conversion or loading purposes, and all the system shall be properly tested according to the updated test cases specifications for each component.

References and bibliography

[1] Critical Software SA – Company. Critical Software, SA, March 2008 <http://www.criticalsoftware.com/company.html>.

[2] Company – Infineon Technologies. Infineon Technologies, January 2008 <http://www.infineon.com/cms/en/corporate/company/index.html>.

[3] Semiconductor device fabrication – Wikipedia, the free encyclopedia. Wikimedia Foundation, December 2007 <http://en.wikipedia.org/wiki/Semiconductor_fabrication>.

[4] Semiconductor Manufacturing Tour. Infrastructure, December 2007 <http://www.infras.com/Tutorial/>.

[5] Ferreira, Helder. EBS Data Model – an overview. Infineon Technologies, January 2006.

[6] Pessoa, Luis. Y.O.D.A. Training Session – Y.O.D.A. Overview. Critical Software, SA, June 2007.

[7] Standard Test Data Format – Wikipedia, the free encyclopedia. Wikimedia Foundation, December 2007 <http://en.wikipedia.org/wiki/Standard_Test_Data_Format>.

[8] Standard Test Data Format (STDF) Specification Version 4. Teradyne, December 2007 <http://etidweb.tamu.edu/cdrom0/image/stdf/spec.pdf>.

[9] Salland Engineering B.V.. Salland Engineering, March 2008 <http://www.salland.com/>.

[10] Galaxy Examinator. Galaxy Semiconductor Solutions, March 2008 <http://www.galaxysemi.com/examinator/support/faq.htm>.

[11] Silva, Daniel. STDF2EXF Converter Software Requirements Specification. Critical Software, SA, March 2008. CSW-EBSDBLGC-2007-SRS-7556.

[12] Casimiro, Hugo and Fraga, Leonardo. EXF Schema Specification. Critical Software, SA, August 2007. CSW-EBSDBLGC-2006-SPC-3961.

[13] Marshall, Casey. PySTDF Blog. March 2008 <http://pystdf.blogspot.com/>.

[14] Pystdf – Google code. Google, March 2008 <http://code.google.com/p/pystdf/>.

[15] SEDana – the practical data analysis tool. Salland Engineering, March 2008 <http://www.salland.com/brochures/SEDana.pdf>.

[16] Spry Software. Spry Software, March 2008 <http://www.sprysoftware.net/Products.shtml>.

[17] C++ – Wikipedia, the free encyclopedia. Wikimedia Foundation, March 2008 <http://en.wikipedia.org/wiki/C%2B%2B>.

[18] PL/SQL – Wikipedia, the free encyclopedia. Wikimedia Foundation, March 2008 <http://en.wikipedia.org/wiki/PL_SQL>.

[19] Silva, Daniel. STDF2EXF Converter Test Case Specification. Critical Software, SA, February 2008. CSW-EBSDBLGC-2007-TCS-7636.

[20] Silva, Daniel. STDF2EXF System Test Report. Critical Software, SA, February 2008. CSW-EBSDBLGC-2007-TSR-9755.

[21] Silva, Daniel. STDF2EXF Regression Testing Specification. Critical Software, SA, February 2008. CSW-EBSDBLGC-2008-TCS-01687.

Deitel, H.M. and Deitel, P.J.. C++ How to Program. New Jersey: Prentice Hall, 1994.

ANNEX A: Project history

Week 1: from 01/10/2007 to 05/10/2007

- EBS DB formation.
- EBS DB documentation reading and source code overview.
- Project website development.

Week 2: from 08/10/2007 to 12/10/2007

- STDF2EXF Converter software requirement specification.
- STDF2EXF Converter test case specification.
- Project kick-off meeting.

Week 3: from 15/10/2007 to 19/10/2007

- STDF2EXF Converter software requirement specification.
- STDF2EXF Converter test case specification.
- EBS DB formation.

Week 4: from 22/10/2007 to 26/10/2007

- STDF2EXF Converter code analysis.
- STDF Loader’s logic migration into the existing converter template.

Week 5: from 29/10/2007 to 02/11/2007

- STDF2EXF Converter coding, compilation and testing.
- Progress meeting.

Week 6: from 05/11/2007 to 09/11/2007

- STDF2EXF Converter coding (standalone mode – work packages reading).
- Progress meeting.

Week 7: from 12/11/2007 to 16/11/2007

- STDF2EXF Converter coding (standalone mode – I/O).

Week 8: from 19/11/2007 to 23/11/2007

- STDF2EXF Converter coding (standalone mode – I/O, conversion and logging).
- Progress meeting.
- Coordination meeting.

Week 9: from 26/11/2007 to 30/11/2007

- STDF2EXF Converter coding (standalone mode – WP/EXP processing).

Week 10: from 03/12/2007 to 07/12/2007

- STDF2EXF Converter coding (standalone mode – first working version and code enhancements).

Week 11: from 10/12/2007 to 14/12/2007

- STDF2EXF Converter code enhancements.
- Internal project presentation preparation.
- Documentation update.
- Progress meeting.

Week 12: from 17/12/2007 to 21/12/2007

- Internal project presentation.
- STDF2EXF Converter testing.
- Documentation update.

Week 13: from 24/12/2007 to 28/12/2007

- Project report writing.
- EBS-DB-LOGIC maintenance.

Week 14: from 31/12/2007 to 04/01/2008

- Project report writing.
- EBS-DB-LOGIC v2.4.1.0 development.
- STDF2EXF Converter testing.
- Progress meeting.

Week 15: from 07/01/2008 to 11/01/2008

- Project report writing.
- EBS-DB-LOGIC v2.4.1.0 development.
- EXF Loader and STDF Loader API study.
- EXF Loader development.

Week 16: from 14/01/2008 to 18/01/2008

- EBS-DB-LOGIC v2.4.1.0 installation testing.
- EXF Loader development.
- Progress meeting.

Week 17: from 21/01/2008 to 25/01/2008

- EXF Loader development.

Week 18: from 28/01/2008 to 01/02/2008

- EXF Loader development.
- Documentation update.
- EBS-DB-LOGIC v2.4.2.0 technical specification and development.

Week 19: from 04/02/2008 to 08/02/2008

- EBS-DB-LOGIC v2.4.2.0 technical specification and development.
- EXF Loader development.

Week 20: from 11/02/2008 to 15/02/2008

- EBS-DB-LOGIC v2.4.2.0 release tasks.
- EXF Loader development.
- Project report writing.

Week 21: from 18/02/2008 to 22/02/2008

- Project report writing.

Week 22: from 25/02/2008 to 29/02/2008

- Project report writing.
- EBS DB regression testing (STDF2EXF Converter and EXF Loader).

Week 23: from 03/03/2008 to 07/03/2008

- Project report writing.
- Deliverables handling.

Regular contact was made by MSN Messenger, e-mail or personal meetings between the author and Prof.ª Ana Paula Rocha, the supervisor at FEUP.