



beyond
payment

TELIUM2 to TELIUM+ framework migration -HOW TO

ICO-OPE-00802

Contents

| | |
|--|-----------|
| 1. Document Information | 5 |
| 1.1 Evolution follow-up | 5 |
| 1.2 Document validity | 5 |
| 1.3 Objective | 5 |
| 1.4 Scope | 5 |
| 1.5 Abbreviations | 5 |
| 1.6 Terminology | 6 |
| 1.7 Documents References | 7 |
| 2. Introduction | 8 |
| 2.1 TELIUM+ framework characteristics | 8 |
| 2.2 TELIUM+ framework usage consequences | 9 |
| 2.3 TELIUM+ framework new logical services | 9 |
| 2.3.1 TELIUM OS_Layer | 9 |
| 2.3.2 TELIUM DLL data new management mode | 10 |
| 3. Be ready to migrate | 12 |
| 3.1 Inventory | 12 |
| 3.1.1 “AppParser” tool | 12 |
| 3.1.2 INGEDEV and development process | 13 |
| 3.1.3 “Inventory” humanly performed | 14 |
| 3.1.3.1 Application compliancy to TELIUM Development Rules | 14 |
| 3.1.3.2 Data sharing between applications | 14 |
| 3.1.3.3 Data management in applications DLL | 14 |
| 3.1.4 “Inventory” result usage | 15 |
| 3.2 “Ready to Migrate” state | 15 |

| | | |
|-------|--------------------------------------|----|
| 3.2.1 | Updates according to deprecated APIs | 15 |
| 3.2.2 | DLL specific case | 16 |

4. T2 and T+ framework packaging and usage 18

| | | |
|-------|---|----|
| 4.1 | New SDK delivery | 18 |
| 4.1.1 | Current SDK delivery | 18 |
| 4.1.2 | SDK delivery new structure | 19 |
| 4.2 | TELIUM + framework in INGEDEV | 20 |
| 4.2.1 | Native TELIUM project creation logic | 20 |
| 4.2.2 | New wizard to support Telium + framework | 21 |
| 4.2.3 | New TELIUM DLLs wizard on TELIUM 2 and TELIUM+ frameworks | 24 |

5. Migration to TELIUM+ framework 25

| | | |
|-------|---|----|
| 5.1 | INGEDEV assistant to TELIUM + framework migration | 25 |
| 5.1.1 | T2 to T+ framework migration assistance | 25 |
| 5.1.2 | Static Libraries sharing between applications migration | 26 |
| 5.2 | Migration to TELIUM + framework without INGEDEV assistant | 26 |
| 5.3 | Migration to TELIUM + framework specific cases | 29 |
| 5.3.1 | CPP project migration to TELIUM + framework | 29 |
| 5.3.2 | TELIUM “vsprintf” specificity | 30 |
| 5.3.3 | TELIUM “RegisterPowerFailure” deprecation specific case | 31 |
| 5.3.4 | Applications targeting all the terminals of TELIUM range | 31 |

Table of figures and tables

| | |
|--|----|
| Table 1 - Follow-up | 5 |
| Table 2 - Validity | 5 |
| Table 3 - Abbreviations | 6 |
| Table 4 - Terminology | 6 |
| Table 5 - Documents references | 7 |
| Table 6 - DLL use cases | 17 |
| Figure 1 – Deprecated API identification by project building | 9 |
| Figure 2 – DLL T2/ T+ Memory usage comparison | 10 |
| Figure 3 – Inventory with “AppParser” | 12 |
| Figure 4 – Deprecated API identification by project building | 13 |
| Figure 5 – Deprecated API inventory thanks to CHM | 13 |
| Figure 6 – Controlling “Ready to Migrate” in INGEDEV | 16 |
| Figure 7 – Current SDK installation directory view | 18 |
| Figure 8 – New SDK installation directory view | 19 |
| Figure 9 – Project creation logic | 21 |
| Figure 10 – Installed TELIUM Packages with framework notion | 22 |
| Figure 11 – TELIUM+ framework project creation | 22 |
| Figure 12 – TELIUM+ project type selection | 23 |
| Figure 13 – TELIUM+ SDK and Add-ons selection | 23 |
| Figure 14 – TELIUM DLL data mode selection | 24 |
| Figure 15 – “sdk30.h” global search and replace | 27 |
| Figure 16 – “sdk30.h” global replace | 27 |
| Figure 17 – “sdk30.h” global replacement detailed preview | 27 |
| Figure 18 – “MigrateToTPLUS.xml” file | 28 |
| Figure 19 – Code source “Refactor” operation | 28 |
| Figure 20 – Eclipse CDT “Refactor” operation | 29 |
| Figure 21 – CPP constraints removed with TELIUM + framework | 30 |

1. Document Information

1.1 Evolution follow-up

| Revision | Type of modification | Author | Date |
|----------|---|------------------|------------|
| 0.1 | Draft version creation | FRIMOUR Jean-Luc | 2013/04/11 |
| 1.0 | Final version | FRIMOUR Jean-Luc | 2013/05/23 |
| 2.0 | Clarifications on: - DLL management - Migration without help of INGEDEV assistant - order of chapter 4 and 5 changed | FRIMOUR Jean-Luc | 2013/06/13 |
| | | | |

Table 1 - Follow-up

1.2 Document validity

| | Name | Function | Signature | Date |
|-------------|-----------------------|-----------------------------|-----------|------------|
| Verified by | FRIMOUR Jean-Luc | PLATFORM Software Architect | | 2013/06/19 |
| Verified by | MENET Yannick | Product Marketing | | 2013/06/19 |
| Approved by | BARTHELEMY Christophe | PLATFORM Product Manager | | 2013/06/19 |

Table 2 - Validity

1.3 Objective

This document presents HOW to migrate a TELIUM T2 framework based project to a TELIUM+ framework project.

1.4 Scope

This document is intended to be provided inside the SDK package.

1.5 Abbreviations

| Abbreviation | Meaning |
|--------------|--|
| THx | Thunder x = name of the core processors , base of TELIUM hardware architecture. |

| | |
|-------------------|---|
| T2 | TELIUM 2 = terminology covering current hardware reference design based on TH2 or TH3 core processor and the associated TELIUM software framework. (=> by extend binary format supported by TH2 or TH3 core based terminals) |
| T+ | TELIUM + = Reviewed software framework, inherited from Telium 2 framework, compatible with T1 and T2 terminals. |
| SDK | Software Development Kit <i>Note: As this terminology is often implicitly associated to TELIUM PLATFORM delivery, take care in using it in other context, it may lead to misunderstanding.</i> |
| SDK Tx | Software Development Kit providing Tx framework and the associated binaries to load in different terminal targets |
| EOL | End of life |
| HW | Hardware |
| SW | Software |
| API (APIs) | Application Programming Interface (s) |

Table 3 - Abbreviations

1.6 Terminology

Specific meaning of some words used in this document, are detailed in the table below.

| Terms | Meaning |
|---------------------------------|--|
| Framework | In this document, define a set of APIs available for applications development. It takes form of: <ul style="list-style-type: none"> Header files (= APIs definition) Different interfaces “libraries” used to build applications according hardware target or building tool chains. |
| Migration | Different actions to perform by developers on an existing TELIUM 2 project to obtain a project compliant with TELIUM+ framework |
| Application | According to context, can refer to: <ul style="list-style-type: none"> Binary executable code file that a TELIUM Operating System can run. A logical entity composed of different binary executable code files that interact together in a way to treat an activity (ex: EMV application payment, loyalty management, advertising...) |
| Legacy applications | Refer to applications build with TELIUM historical APIs set until TELIUM+ framework introduction |
| PLATFORM | In this document when not associated specifically to software or hardware, name the couple hardware/software on which an application will run. |
| Binaries | Executable files that can be loaded in terminal. They format may differ according to terminal hardware target and it contents: <ul style="list-style-type: none"> Authentication/identification information Binary executable code |
| Software Development Kit | Software Development Kit (or Tools Kit) allows creating an application. It contents: <ul style="list-style-type: none"> One or several frameworks and their documentation (CHM, Samples,...) One or several set of terminal loadable binaries according to hardware target and to the provided framework Note: <ul style="list-style-type: none"> Binaries could support more than one framework at a time Binaries functional sharing may be different according to hardware targets. |

Table 4 - Terminology

1.7 Documents References

| Category | Title | Filename or reference |
|----------|--------------------------------------|---|
| INGENICO | Telium 2 Framework Ingedev help file | SDK_TELIUM_vvrrxx.chm (provided by SKD setup) |
| INGENICO | Telium+ Framework Ingedev help file | SDK_TELIUMPLUS_vvrrxx.chm (provided by SKD setup) |
| INGENICO | Telium development rules | ICO-OPE-00156-Vx Telium development rules.pdf (provided by SKD setup) |

Table 5 – Documents references

2. Introduction

In order to improve TELIUM Platform, a survey has been proposed to INGENICO Regions and to some of their VARs about the TELIUM SDK and Add-ons APIs usage. The different results analyse has shown a significant part of APIs not used.

According to these results and to anticipate future PCI V4 compliance, it has been decided to propose to developers a reviewed TELIUM framework named “**TELIUM+ framework**”.

2.1 TELIUM+ framework characteristics

The main technical characteristics of TELIUM+ framework are:

- A subset of **unmodified** APIs inherited from TELIUM 2 framework
 - Except for a very small list of exceptions, 100% of the functional behavior is maintained
 - The list of the maintained APIs has been established **based** on a survey organized with TELIUM PLATFORM customers and based on the following criteria:
 - API is old, non well documented, unused or rarely used
 - API is redundant with another API functionally richer.
 - APIs that could present issues face to new security constraints
 - Command managed by “fioc!” API which is old, none well documented, unused or rarely used or redundant with a more abstracted PLATFORM API.
 - **API is provided by some Add-ons to SDK declared end-of-life.**(ex: U32 Migration Add-ons = TELICAPT and EMV Migration Layer)
- A subset of **renamed** APIs present in TELIUM 2 framework.
 - The renaming causes are:
 - The APIs are in conflict with standard “C” library and forbid to manage easily projects in “C++”.
 - The APIs was historically based on standard and their usage has required INGENICO evolutions or customizations not compatible with current existing off selves “open sources” standard APIs, which may be useful to introduce new features. (ex: IP V6).
 - their naming not currently exposed in English
 - their naming does not reflect their functional perimeter
- A set of APIs providing **new logical services**
 - Current internal services used in PLATFORM components but not published for applications usage. (ex: RemoveCardWithBeep())
 - New abstraction of Operating SYSTEM features
 - Less dependant of embedded terminal target Operating SYSTEM (= easy to port in WIN32 for tests purpose or other)
 - New features to comply with future security constraints
 - New mechanisms to manage DLL in TELIUM + framework context according to new security constrains.
 - *Note: To easier migration from TELIUM2 to TELIUM+ framework, some new logical services will also be delivered in TELIUM 2 framework.*

2.3.2 TELIUM DLL data new management mode

The current TELIUM mechanism has been enhanced to respond to the following requirements:

- Providing a DLL with a specific data context, linked with the application calling the DLL.
- Dedicated mechanism to share data between applications through DLL with a logical identification.
- Simplest mechanism to Load/Unload the DLLs with capability to call a treatment during these two operations.
- DLL calling mechanism improvement.

For developer, the consequences are:

- The developer has the choice on how the data of its DLL is managed
 - Data can be implicitly shared between applications. (current existing mode)
 - One instance of data of DLL exists per application calling the DLL = no shared data between applications through DLL usage. (new created mode => see figure below)
- No need to call “InitVar” anymore (= automatically done by the new loading mechanism)
- In paying attention to backward compatibility, new call mechanism can be used with performance call improvement and a significant memory code/data or data wins.
- In using the new TELIUM DLL data mode, shared data between applications are better well identified: this anticipates support of PCI V4 security constraints. However a well written DLL without using new data mode, may lead to the same result without the constraints of memory consumption increase.

The following figure illustrates the difference between the two data modes of TELIUM DLLs

- Considering 2 applications ‘APP1’ and ‘APP2’ calling the same DLL
- With implicit data sharing mode, data ‘A’ is unique, its evolution could be dependant of calls of ‘APP1’ and of ‘APP2’
- With no data sharing mode, data ‘A’ is duplicate.

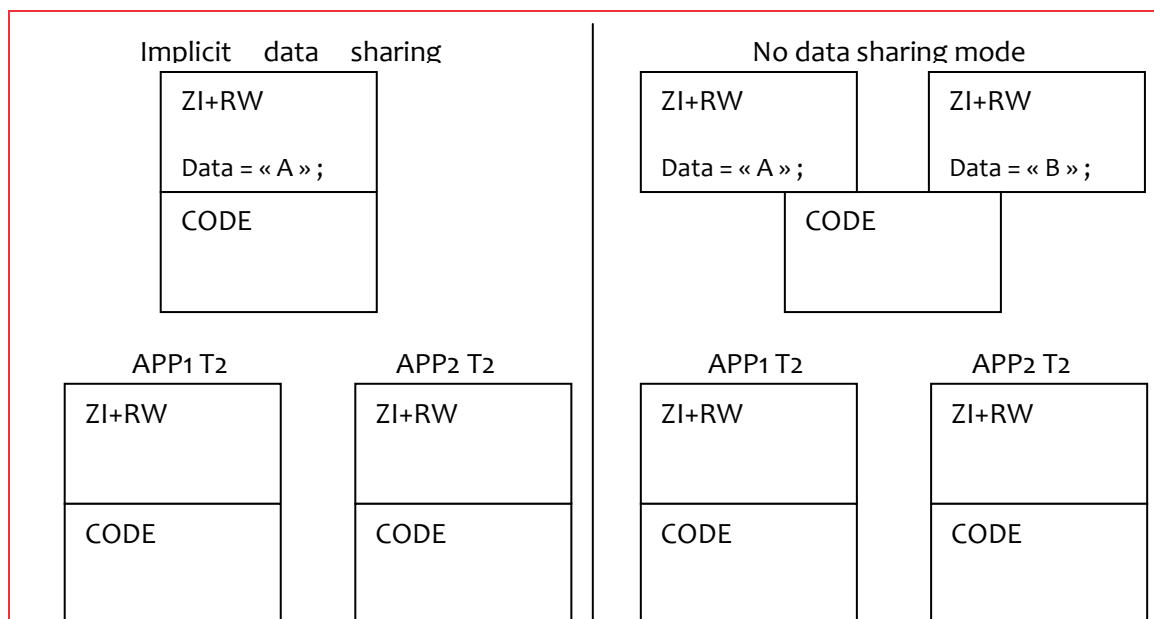



Figure 2 – DLL T2/ T+ Memory usage comparison

Even if this new mechanism is safer, pay attention on the fact that:

| | | |
|---|---|---------------|
|  | TELIUM2 to TELIUM+ framework migration -HOW TO | ICO-OPE-00802 |
|---|---|---------------|

- Each instantiation of the DLL will need a specific pool of memory.
- Memory footprint when using TELIUM DLL no sharing data mode is significantly increased.
- Do not use superfluous amount of memory => evaluate your real needs.

3. Be ready to migrate

Three steps can be considered in migration to TELIUM+ framework.

- TELIUM 2 project “Inventory”
- TELIUM 2 project “Ready to migrate”
- “Migration” to TELIUM+ framework

3.1 Inventory

The “**Inventory**” operation main objective is to evaluate application (and its sub-components) migration to TELIUM + framework capacities.

The input to perform “**Inventory**” is a TELIUM 2 project.

Since SDK 9.8.0 delivery, developers can anticipate and prepare TELIUM+ framework migration thanks to:

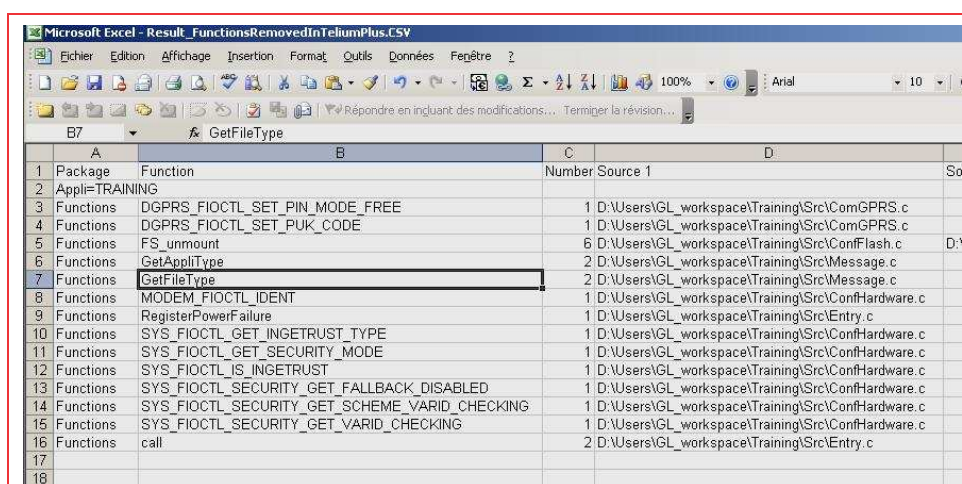
- “**AppParser**” tool
- “**INGEDEV**”
- “**CHM**” document delivered in SDK package

3.1.1 “AppParser” tool

“**AppParser**” tool is provided in the “**tools|AppParser**” SDK install directory. It can be used to identify the deprecated APIs used in an application by parsing the source code.

After having install the tool, refer to user’s manual to know:

- How to use the tool
- Which option to select to obtain the list of source files contenting APIs lacking in TELIUM+ framework and also the list of these deprecated used APIs.



| A | B | C | D |
|----------------|---|--------|---|
| Package | Function | Number | Source 1 |
| Appli=TRAINING | | | |
| Functions | DGPRS_FIOCTL_SET_PIN_MODE_FREE | 1 | D:\Users\GL_workspace\Training\Src\ComGPRS.c |
| Functions | DGPRS_FIOCTL_SET_PUK_CODE | 1 | D:\Users\GL_workspace\Training\Src\ComGPRS.c |
| Functions | FS_unmount | 6 | D:\Users\GL_workspace\Training\Src\ConfFlash.c |
| Functions | GetApplType | 2 | D:\Users\GL_workspace\Training\Src\Message.c |
| Functions | GetFileType | 2 | D:\Users\GL_workspace\Training\Src\Message.c |
| Functions | MODEM_FIOCTL_IDENT | 1 | D:\Users\GL_workspace\Training\Src\ConfHardware.c |
| Functions | RegisterPowerFailure | 1 | D:\Users\GL_workspace\Training\Src\Entry.c |
| Functions | SYS_FIOCTL_GET_INGETRUST_TYPE | 1 | D:\Users\GL_workspace\Training\Src\ConfHardware.c |
| Functions | SYS_FIOCTL_GET_SECURITY_MODE | 1 | D:\Users\GL_workspace\Training\Src\ConfHardware.c |
| Functions | SYS_FIOCTL_IS_INGETRUST | 1 | D:\Users\GL_workspace\Training\Src\ConfHardware.c |
| Functions | SYS_FIOCTL_SECURITY_GET_FALLBACK_DISABLED | 1 | D:\Users\GL_workspace\Training\Src\ConfHardware.c |
| Functions | SYS_FIOCTL_SECURITY_GET_SCHEME_VARID_CHECKING | 1 | D:\Users\GL_workspace\Training\Src\ConfHardware.c |
| Functions | SYS_FIOCTL_SECURITY_GET_VARID_CHECKING | 1 | D:\Users\GL_workspace\Training\Src\ConfHardware.c |
| Functions | call | 2 | D:\Users\GL_workspace\Training\Src\Entry.c |

Figure 3 – Inventory with “AppParser”

Note: The figure below illustrates the obtained result after a parsing of “TRAINING” project: “AppParser NS”: The result is an EXCEL file.

3.1.2 INGEDEV and development process

In changing the SDK reference in an existing project, in using **“Manage TELIUM package”** menu and selecting a SDK version > 9.8, the project rebuilding will allow identifying deprecated functions used.

- Deprecated APIs appear with a compilation error. (as illustrated in figure below)

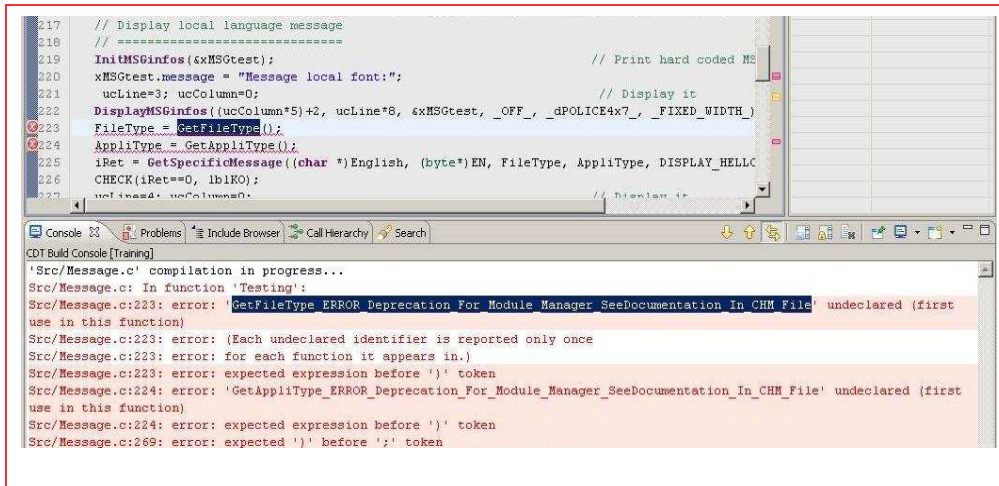


Figure 4 – Deprecated API identification by project building

- Having a look in CHM document will inform on the detailed status of the API:
 - Which API (s) or which feature to use in place (= in figure example ObjectGetInfo must be used in place)
 - Which define to set to build the project without changes.(= to transform error in warning)
- The same information can be obtained more quickly but in a less formalized way, thank to a right clicking and selecting **“Open declaration”** or thank to key **“F3”** hitting: the include file contenting the declaration of deprecated API appears.
- This operation must be done on each **“Build Configurations”** when conditional building is applied on some parts of source files.

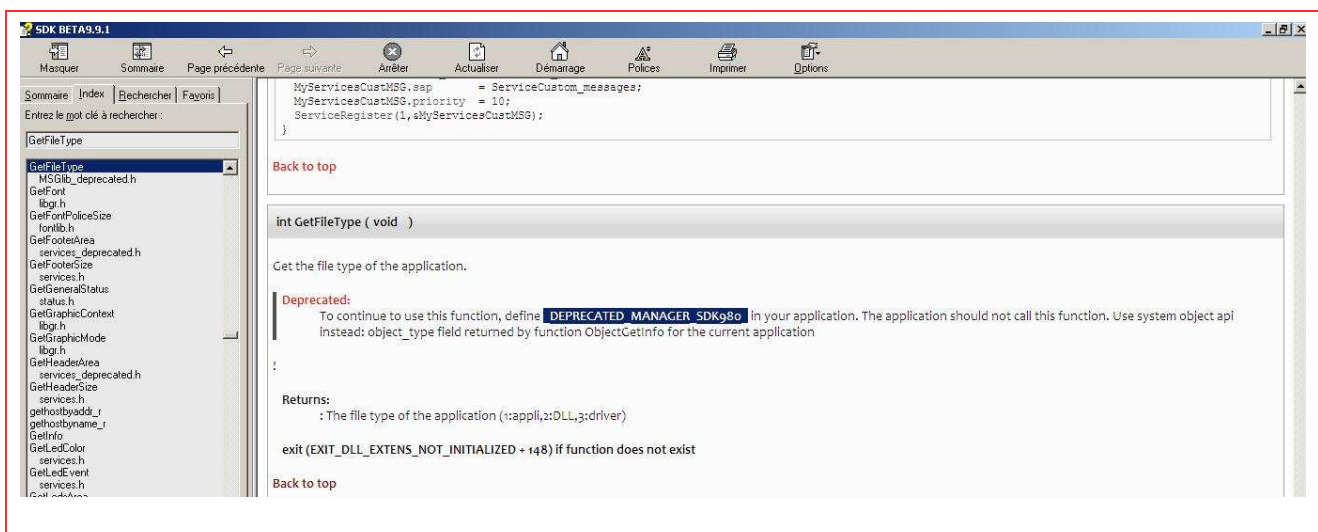


Figure 5 – Deprecated API inventory thanks to CHM

3.1.3 “Inventory” humanly performed

To anticipate future PCI V4 compliance and to migrate DLL, “**Inventory**” perimeter must enlarge and requires some little extra work humanly performed to better know the applications characteristics: Analyzes done on applications architecture or detailed specifications, on source files, etc ... will help to establish some figures allowing to determine when it will be needed, the application compliancy to new security PCI V4 requirements.

According to today security status, a part of security constraints can be taken in account by anticipation during application evolution.

A main point to investigate and that can be summarized by “Data segregation” between applications. It concerns:

- Application compliancy to “**TELIUM Development Rules**” (refer to user manual provided in CHM)
- The way the application shares data between together.
- How data in DLL are managed.

3.1.3.1 Application compliancy to TELIUM Development Rules

Concerning this point refer to user manual provided in CHM.

One very important point to highlight is including files usage:

- If the included files of “**sdk30.h**” have been used in place of direct inclusion of “**sdk30.h**”, the automatic migration assistance provided by INGEDEV may not reach objectives.

3.1.3.2 Data sharing between applications

Points to put in the inventory:


- Is my application sharing data with other applications?
- Which is the link between these applications?
 - A logical module of a main application (created only to easier to maintenance).
 - A real independent application (payment) with its own specificity and security requirement...
- What are technical ways used to shared data? (FILE, common memory , message service mechanism ...)
- Are sharing data limited to only useful information?
- Are my applications using synchronization mechanisms between together? (semaphore, message services...)
- Are there some mechanisms that prevent against no authorized applications to manipulate these shared information?

3.1.3.3 Data management in applications DLL

Currently, in a native TELIUM DLL, by design, all DATA are shared by default.

Points to put in inventory concerning DLL are:

- Did I use implicit data sharing mechanism of native TELIUM DLLs to share information between applications?
- Are the DLL treatments re-entrant?
- Are there mechanisms preventing against more than one application using the DLL at a time?
- Does my DLL really need data for itself?

| | | |
|---|---|---------------|
|  | TELIUM2 to TELIUM+ framework migration -HOW TO | ICO-OPE-00802 |
|---|---|---------------|

- Can the DLL be easily modified to only work on provided information coming from application calling it?
- Are sharing data limited to only useful information?
- Are there some mechanisms that prevent against no authorized application to use/call the DLL?

3.1.4 “Inventory” result usage

At this step, having an inventory of the changes to do, the user can decide to:

- Go on next step => make its project ready to migrate to TELIUM+ framework.
- Delay this operation and work on its project as is

To work on the project, without rework, the user must introduce different **“DEFINE”** accorded to deprecated APIs used, thanks to **“Compiler”** -> **“User symbols”**. This operation must be done on each **“Build Configurations”**. This done:

- All errors are transformed in simple warning.
- Binaries obtained are unchanged

Note: According to the example shown in [Figure 4 – Deprecated API identification](#), a define `_DEPRECATED_MANAGER_SDK980_`, obtained thanks to CHM (illustrated in [Figure 5 – Deprecated API inventory thanks to CHM](#)) will allow to work without changes on source code.

Note: If all “Build Configurations” need the same set of DEFINES, you can set them on all existing configurations with a single operation:

- Select in “Build Configurations” : [All configurations]
- Introduce define in “Compiler” -> “User symbols”.

3.2 “Ready to Migrate” state

“Ready to Migrate” is a state that must reach a TELIUM 2 framework based project to allow an easy migration to TELIUM+ framework: It can be considered as the first mandatory step of the migration although the result is not TELIUM+ framework project but TELIUM 2 framework project .

This step consist in modifications that the developer must do itself on the software according to the inventory result:

- There is nothing to do on some projects
- Huge work must be done to adapt the projects in particular if DATA structure organization has neglected or to anticipate PCI v4 security requirements taking in account.

Three aspects have to be considered:

- Project changes according to deprecated APIs
- DLL migration to TELIUM+ framework
- DATA management in the application and its linked components

When this step will be reached, “conversion” to TELIUM+ framework can be engaged or postponed to another application software evolution.

3.2.1 Updates according to deprecated APIs

To perform the operation, select a TELIUM 2 framework project on which no “deprecated” defines have been set.

When the “Inventory” has been done, the operation to obtain a “clean” TELIUM 2 framework based project can begin, it consists in:

- Analyzing if the treatments where deprecated APIs are found, continue to be functionally relevant in particular when application is old and may run on multiple target terminals.
- Removing in the source code all the deprecated APIs reported by the inventory
- Adapting the source code where there are removed treatments in adding suggested APIs / solution found in CHM file.
- Each **“Build configurations”** rebuilding
- Verifying that functionally the changes are valid in targeted terminals. This control must be performed also on each possible build configurations.

Note: According to survey result, most existing recent projects will need to change less than 20 APIs. Therefore, there is no need for a complex tool that would automate a limited number of changes.

To ensure that the project has reached **“Ready to Migrate”** state, verify that:

- No define remains present to manage APIs deprecation thanks to a quick look in **“Compiler”** -> **“User symbols”** of each **“Build configurations”**
- The absence of deprecated warning in **“Console”** or **“Problems”** windows.

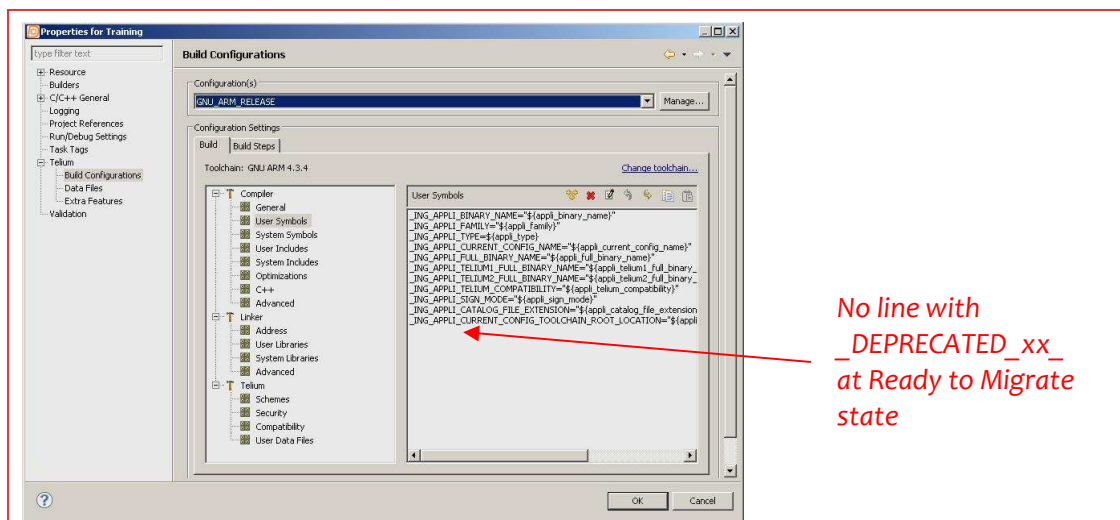


Figure 6 – Controlling “Ready to Migrate” in INGEDEV

As migration can be considered project by project, the applications and their associated DLLs can be considered separately.

TELIUM Static Library Project must be modified to be **“Ready to Migrate”** once an application using it, must comply with **“Ready to Migrate”** state.

3.2.2 DLL specific case

According to “Inventory” result and to use TELIUM DLL new data mode, some modifications have to be done an existing TELIUM DLL.

| Use cases | Modifications to do |
|---|---------------------|
| Implicit data sharing mechanism is not used | Nothing to do |

| | |
|---|---|
| Implicit mechanism used to share information between applications | Modify DLL in using new shared memory mechanism |
| DLL global/static data are used to manage DLL state/behaviour etc.. | Modify DLL in using new shared memory mechanism |
| DLL only work with data provided by the application | Nothing to do |

Table 6 - DLL use cases

Even if there nothing to do:

- Verify that all declared data are useful
- Verify that arrays sizes margin limit are coherent with the usage peak
- Privilege malloc/free usage to limit the data size areas of the DLL (= the possible mapping areas taking benefits of “fast switch context” are limited and constrains by current legacy existing areas and limits).
- Estimate the new memory footprint that the DLL no sharing data mode usage will imply on each targeted terminal using these configurations.

***Note:** Remember that TELIUM application Heap management asks for memory to SYSTEM pool when there is no space in local heap and never retribute to SYSTEM pool when application “free” memory: it’s kept in application to quick up next allocation. So it means that the heap size will reach the application maximum need that allows the application to run whatever the conditions. In consequence to optimize memory usage in terminal, separate data of the application in different allocated memory spaces that are not used simultaneously.*

4. T2 and T+ framework packaging and usage

This chapter describes customers' TELIUM frameworks views.

The framework notion will first appear to applications developers when they have to:

- Create and build an application or sub-project thank to INGEDEV.
- Select the software configuration in SDK deliveries to load its terminal to test and qualify its developed application
- Prepare applications package to feed the TMS server.

4.1 New SDK delivery

4.1.1 Current SDK delivery

Currently the T2 framework is delivered in an auto-install program called **"SDK Telium x.y.x setup.exe"** communally named **"SDK"**.

After installation, the following installation directory is created, its root contents **"SDKDescriptor"** files (used by **INGEDEV** to set the default properties of the project to manage) and different directories:

- **"Component"** = software configuration to load in terminal, sorted by functional modules.
- **"Documents"** = different technical documents + the T2 framework "HELP" file document (.CHM)
- **"SDK"** = framework APIs header included files and the corresponding libraries delivered per building TOOLS Chains.
- **"TOOLS"** = TELIUM WIN32/JAVA components used by **INGEDEV** to manage **"Screen Resource management"** + different tools to manipulate **"Fonts"** **"images"** ...

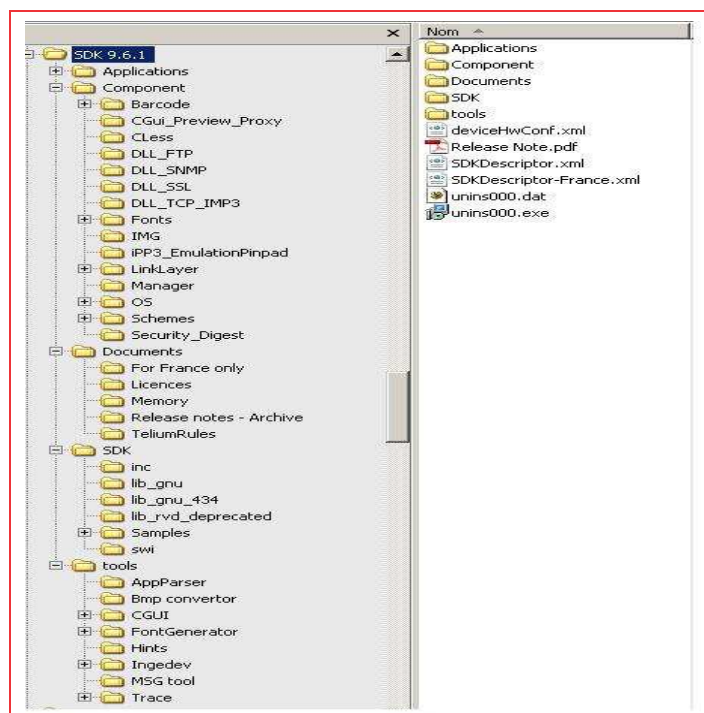


Figure 7 – Current SDK installation directory view

4.1.2 SDK delivery new structure

The new SDK setup install program will put in place some new directories and configuration files in maintaining existing ones for backward compatibility.

- Framework support is given by **“SDK framework descriptor”** file presence at root of the install directory.
 - **“SDKDescriptor.xml”** presence = framework TELIUM 2 supported
 - **“SDKTPLUSDescriptor.xml”** presence = framework TELIUM+ supported
- Each present framework will have its own directory in which can be found includes, libraries to build application, DLL,...
 - **“SDK”** directory => all includes and libraries per tool chain are provided to build framework TELIUM 2 projects. (=same organization as in previous SDK)
 - **“SDKTPLUS”** directory => all includes and libraries are provided to build framework TELIUM+ projects. The directory structure and contents are organized according to standard best practices in offering better evolution flexibility. Information organized per family hardware target and per tool chain:
 - **“Includes files”** are delivered per tool chains
 - **“libraries”** objects are delivered per tool chains

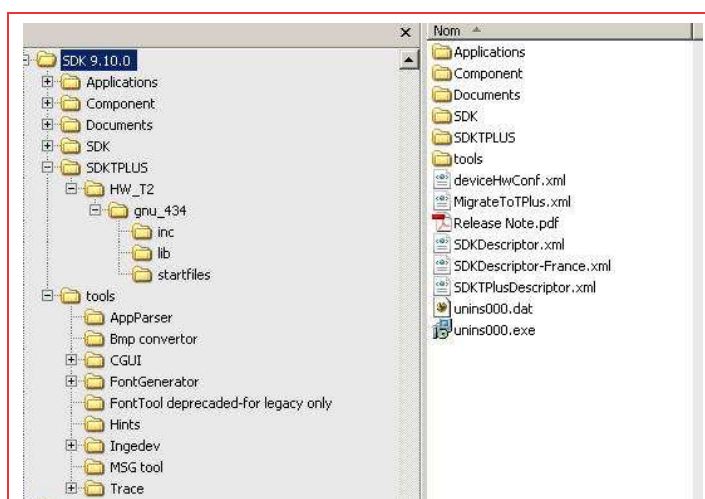


Figure 8 – New SDK installation directory view

- **“MigrateToTPlus.XML”** delivers the list of APIs, ‘C’ elements ,... that will be automatically migrated by INGEDEV migration assistant tool.
- Currently in SDK 9.10, no other evolution has been done in the other provided directories.

Note: According to software evolutions and status (end-of-life), business constraints, it can be decided to deliver SDK install setup programs contenting only one framework. This will not affect the general structure of the delivery: absence of SDK framework descriptor coupled with the corresponding SDK directory.

Note: The TELIUM2 framework descriptor syntax has not be enhanced to allow usage of T2 SDK with an old INGEDEV version.

Note: The TELIUM+ framework descriptor syntax is only recognized by an INGEDEV version >= 7.20.

4.2 TELIUM + framework in INGEDEV

When a developer will want to create projects, he will be facing to different notions: some of them will be new:

- SDK version he has to install or provided by default
- Components to develop (Application, library dynamic or static) and its specificities (GOAL, other components dependencies,...)
- Framework to use (T2 or T+)
- Functional configuration building of the component
- Generation tool chain and hardware targets to support by the component

Note: The delivery of some new notions will be progressive. (= not all provided in INGEDEV version >= 7.20)

4.2.1 Native TELIUM project creation logic

Native project creation under INGEDEV can be seen giving responses to a large list of questions

- Starting with general descriptions of the project
- Up to give detailed description of software to load in specific TELIUM target terminals.

Note: “Native TELIUM project” does not include elements related to INCENDO framework as these components are in practice just some specific native TELIUM applications.

The following figure illustrates step by step the information that the developer will have to provide to create its project bases.

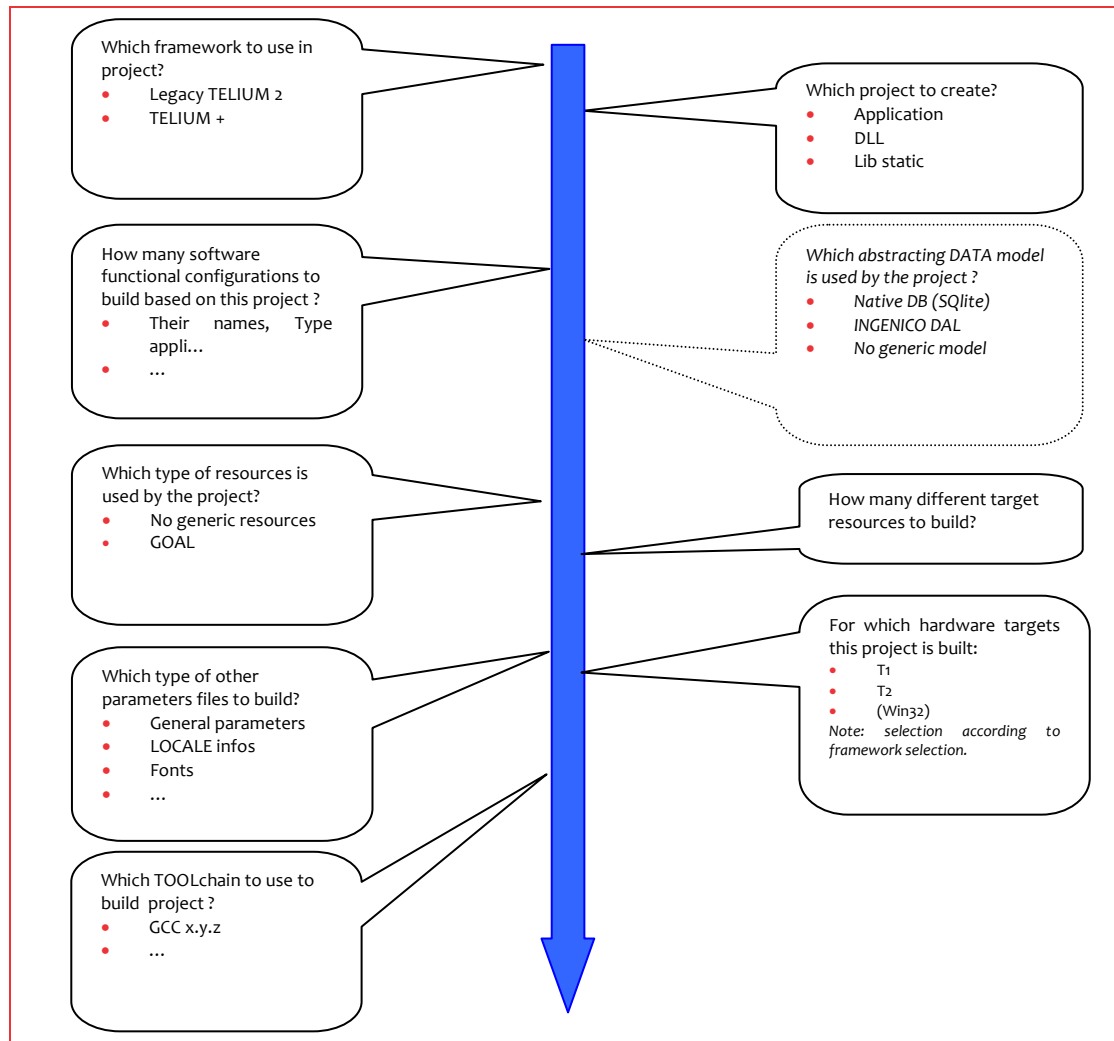


Figure 9 – Project creation logic

4.2.2 New wizard to support Telium + framework

The TELIUM wizard project has been modified to introduce the framework selection: this choice is proposed according to the framework present in the different “**SDK Descriptors**” declared by the developer thanks to “**Ingedev->Telium-> Installed Telium Package**”.

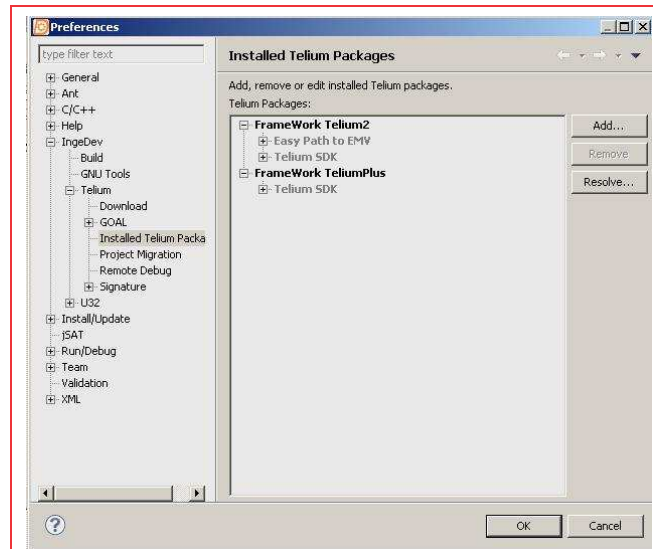


Figure 10 – Installed TELIUM Packages with framework notion

The general project creation has been reviewed to easier future evolution introduction (as DATA description wizard, LOCALE parameter declaration wizard ...).

According to the different frameworks SDK first introduction planning, it will be possible to create projects based on:

- TELIUM 2 framework
- TELIUM+ framework.

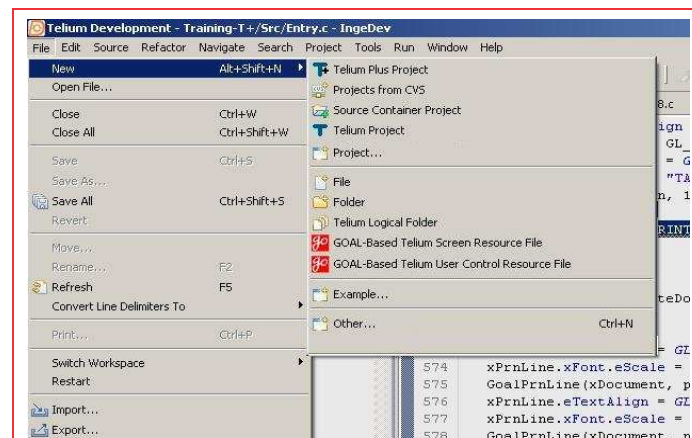


Figure 11 – TELIUM+ framework project creation

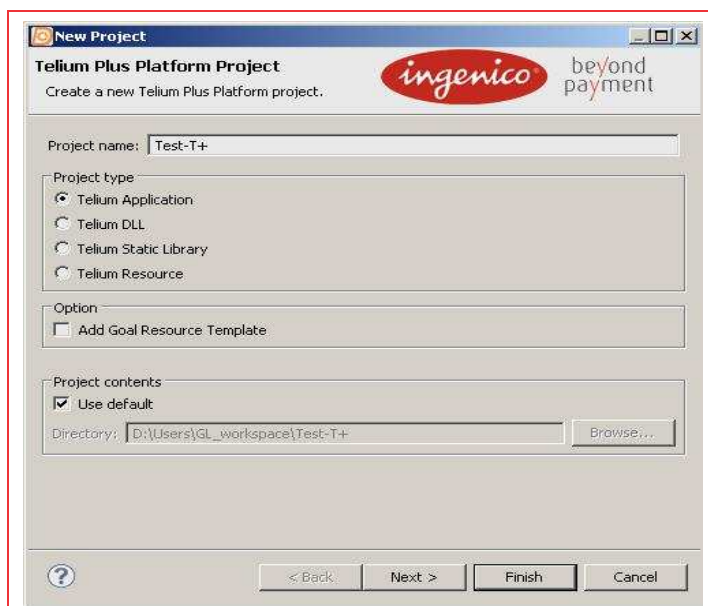


Figure 12 – TELIUM+ project type selection

The **INGEDEV** interface will hide all worry concerning default libraries choice according to hardware targets and tools chains. However manual choice is maintained to allow customization by expert developers.

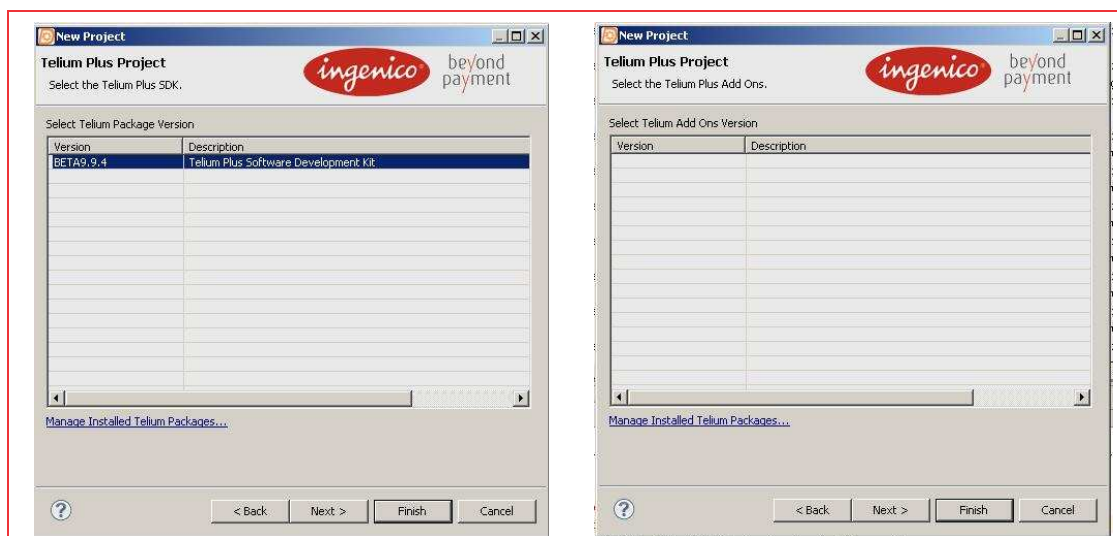


Figure 13 – TELIUM+ SDK and Add-ons selection

The current project wizard is enhanced to better meet the users' needs and to respond to some requirement asked since a long time:

- Improve different functional targets deliveries based on same source files and unique project.
- Allow to create derivate applications in which only **“screen resources”**, printing or locale information are different.
- Add capabilities to manage common configuration on different linked projects deliveries (ex: RELEASE, DEBUG, ...)

4.2.3 New TELIUM DLLs wizard on TELIUM 2 and TELIUM+ frameworks

Whatever the framework, the selection of the TELIUM DLL data mode is available in **“Build Configurations”** -> **“Linker”** -> **“Address”** properties, see figure below:

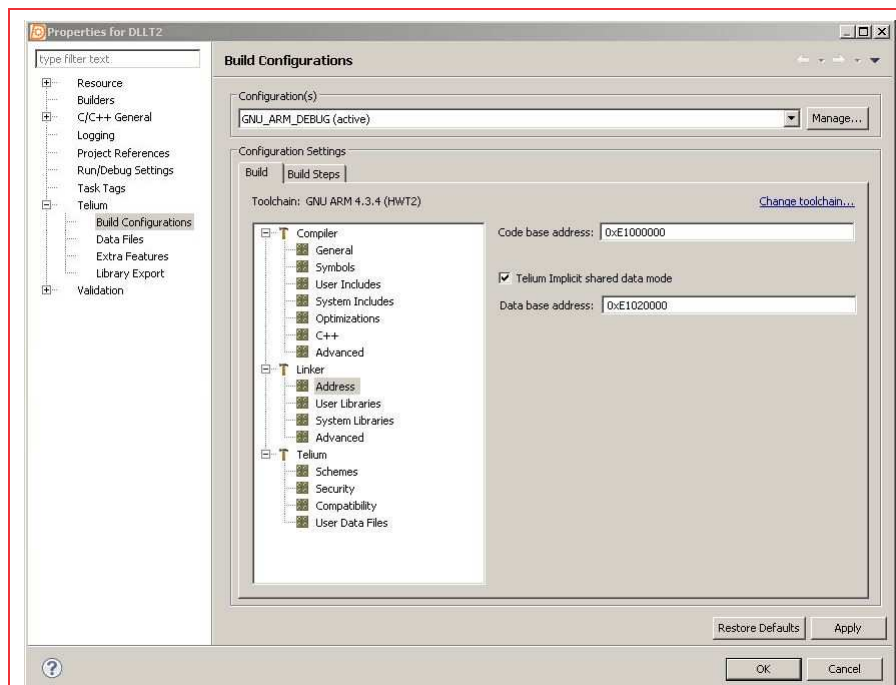


Figure 14 – TELIUM DLL data mode selection

Each TELIUM DLL data mode is associated to specific memory region mapping capabilities:

- **“TELIUM implicit shared data mode”** selection allows putting **“data”** closed to **“code”**. According to your region, a range has been attributed to locate your DLLs in the TELIUM DLL global range (0xE1000000- 0xEFFFFFFF).
- When the new mode is selected, checked box **“TELIUM implicit shared data mode”** not activated, the **“data”** have to be located in a specific range (0x01A00000-0x01FEFFFF) in the first 32 Mega of addressing space.
 - The location in the first 32 Mega guaranty to obtain the same execution performances of DLL whatever their data mode.
 - The area limitation to less than to 2 Mega bytes to locate all the DLL based on new data mode result from compatibility constraints (= current predefined already used areas in this space that cannot be changed or moved).
 - All the DLLs based on this data mode and which have to be present in a terminal for a given configuration, must have a reserved area not overlapping the other ones.

According to the selected framework, when a new project is created, the TELIUM DLL wizards propose different default values on **“TELIUM implicit shared data mode”** and on the associated field **“data”**:

- On TELIUM 2 framework **“TELIUM implicit shared data mode”** is activated by default.
- On TELIUM + framework **“TELIUM implicit shared data mode”** is not activated by default.

5. Migration to TELIUM+ framework

To consider this step your project(s) must be in “Ready to Migrate” state.

This operation is divided in to step:

- Automatic migration phase taken in charge by INGEDEV
- Manual rework on the source code (essentially to do inside DLL project and source code)

5.1 INGEDEV assistant to TELIUM + framework migration

TELIUM project not currently managed thanks to INGEDEV project interface can not take benefit of the migration tool assistance.

So non INGEDEV user must take advantage of the migration to organize their projects building thank to INGEDEV.

5.1.1 T2 to T+ framework migration assistance

INGEDEV will also provide some assistance to migrate application written with T2 framework to T+ framework: when opening a project written with T2 framework, an installed **INGEDEV** with T+ framework capabilities will propose to developer to automate most part of the migration to T+ framework. If it is selected by the developer, **INGEDEV** will automatically:

- Replace the default SYTEM includes and Libraries of the projects.
 - Manually added includes or libraries are unchanged and must be adjusted by user
 - Default SYSTEM includes and libraries, removed by users are not treated and users must reintroduce them manually if needed.
- Update project information
- Rename in the project files (sources and included headers) functions, define, types, ... whose name has been modified between T2 and T+ framework.
- Propose to migrate linked projects.

After migration the project is no more compatible and with a previous **INGEDEV** version.

INGEDEV will propose cyclically to migrate open projects related to applications written with T2 framework.

Refer to INGEDEV CHM for detailed explanations and ways to proceed.

Note: The full delivery of all features, listed previously, will be provided after some INGEDEV incremental release updates.

Note: As in TELIUM+ framework, TELIUM APIs conflicting with standard C library have been renamed, take care to treat all warnings “warning: implicit declaration of function 'xxxxx'” related to standard “C” library functions because no linking errors “undefined reference to `xxxxx'” will append: the current GCC4.3.4 tools chain delivers a standard “C” library no targeting specifically TELIUM. Currently there is no planned work about GCC tools chain new version evolution.

5.1.2 Static Libraries sharing between applications migration

If static libraries are imported between applications, to share common source code / objects, all the concerned applications will have to be migrated simultaneously because a static library must be based on the same framework than the application calling it.

Framework mixing between an application and its static libraries, leads to link errors (unresolved symbols).

If static libraries embed PLATFORM objects/libraries, the framework mixing between application and its static libraries, may be hidden: case of **“no link error”** reported may append.

5.2 Migration to TELIUM + framework without INGEDEV assistant

However INGEDEV provides an assistant to migrate, this operation can easily be done manually.

To start this manual operation, this initial state is:

- The installed version of INGEDEV is > 7.20
- The minimum version of TELIUM SDK package, managing T2 framework, is the version embedded in INGEDEV 7.20 install package.
- All operations to reach **“Ready to migrate”** state have been done on the selected project
- The project result has been verified on a terminal target

Follow the list of operations that have to be done under INGEDEV to manually migrate:

- Install Telium SDK packages corresponding to T+ framework.
- Create a new TELIUM+ project (= the original project is maintained as reference)
- Add all the specific defines, links with others TELIUM project, libraries, etc ...
 - Note that dependant projects must be migrated first.
- In INGEDEV **“Navigator”** view, copy the sources code of the original project and paste them in the **“Src”** directory of the Telium + created project.
- Proceed same way to copy the include files from **“Inc”** to **“Inc”** directory of the Telium + created project.
- In switching to INGEDEV **“TELIUM Explorer”** view, add all the sources files
- If you have followed the **“TELIUM development rules”**, you have only included **“sdk30.h”** in your source files, so you have to first rename **“sdk30.h”** by **“sdk_tplus.h”**.
 - Select Project in TELIUM Explorer view
 - Search **“sdk30.h”** in **“Inclosing project”** and select **“Replace”**

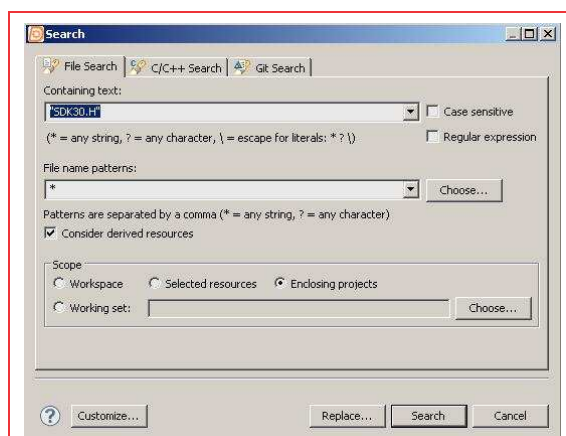


Figure 15 – “sdk30.h” global search and replace

- Set the replacement string to “**sdk_plus.h**”

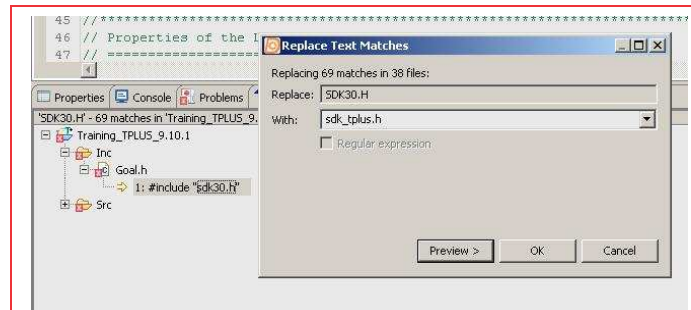


Figure 16 – “sdk30.h” global replace

- It’s possible to verify each detailed changes that will be done in selecting “**Preview**”. Proceed to replacement by selecting “**Ok**”.

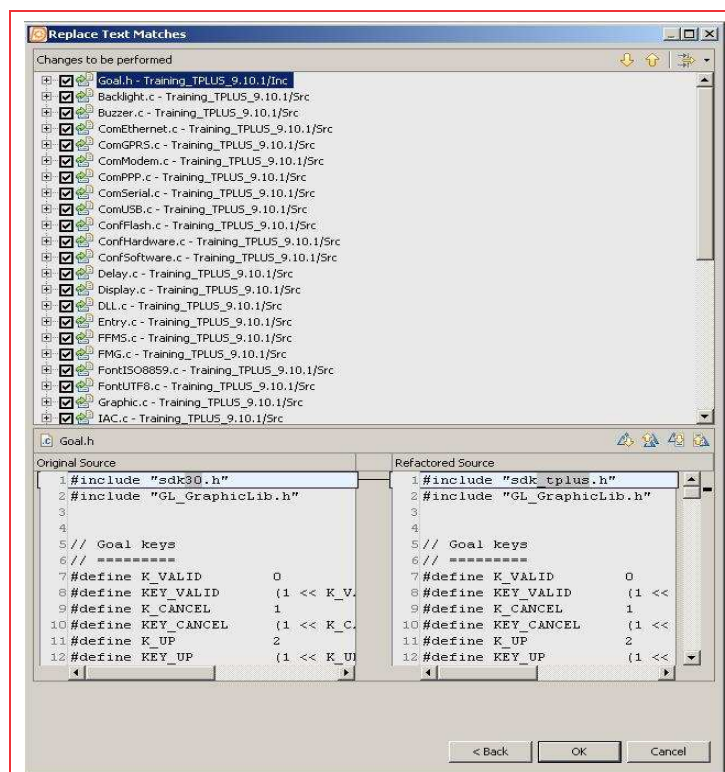


Figure 17 – “sdk30.h” global replacement detailed preview

- Build the list of the functions to rename in using “**AppParser**”:
 - In a DOS prompt: in the installation directory of “**AppParser**” , run this tool with the option RN: “**C:\Program Files\TeliumSDK\AppParser>AppParser.exe RN**”
 - If tool has not been used previously, first create/update “**ListDir.ini**” file (refer to “**AppParser**” user’s manual).
 - Do not add other option; “**RN**” must be used alone.
 - “**Result_FunctionsRenamedInTeliumPlus.CSV**” contents the list of T2 framework functions to rename during migration process.

- At this step, perform **“Built”** operation and replace function name according to functions identified in **“Result_FunctionsRenamedInTeliumPlus.CSV”**: all renamed APIs appearing also in warning like **warning: implicit declaration of function 'xxxxxx'**. The replacement name to **'xxxxxx'** can be found in **“MigrateToTPlus.xml”** file (located at root of the SDK install directory).

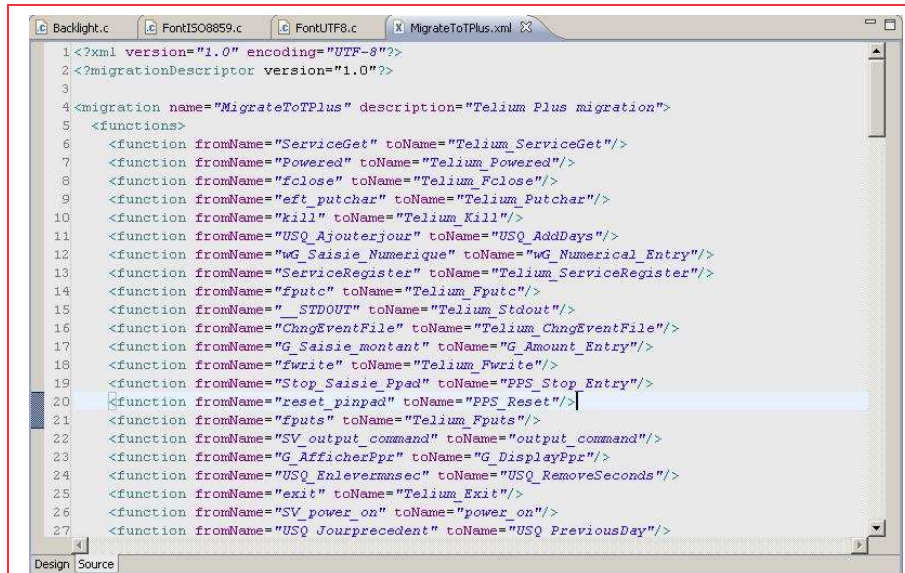


Figure 18 – “MigrateToTPLUS.xml” file

- To perform this operation, privilege Eclipse CDT **“Refactor”** feature usage:
 - Select the API to change, right click and select **“Refactor”** -> **“Rename”** (**“Alt+Shift+R”** keys combinations give the same result), see explanations and pictures below:

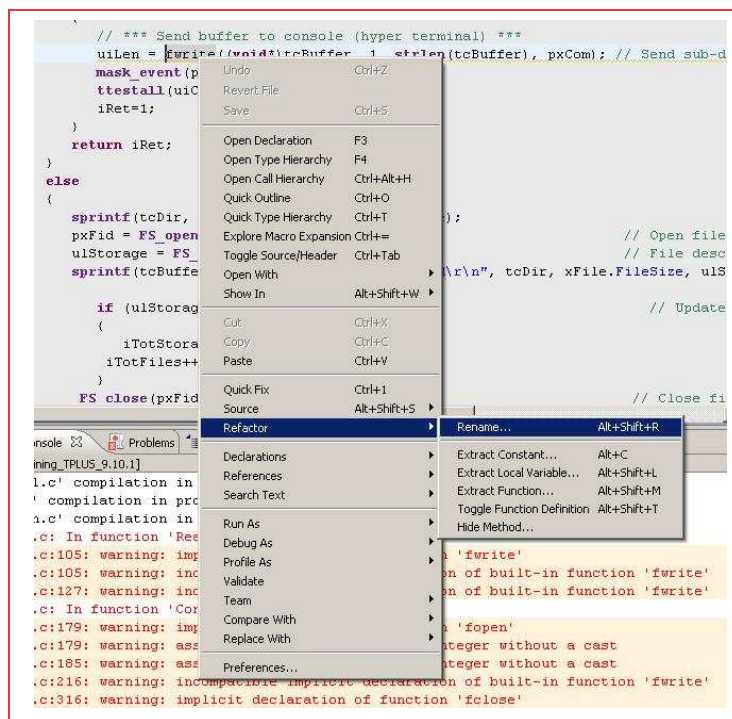


Figure 19 – Code source “Refactor” operation

- The text is highlighted, modify the API name and type « **Enter** ».

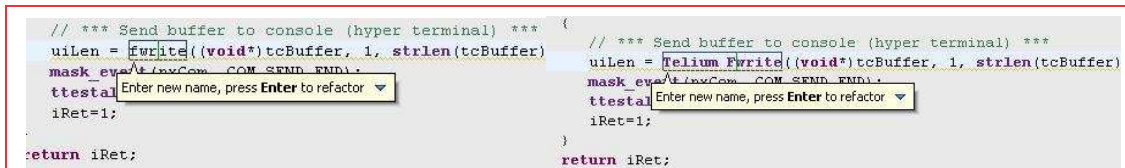


Figure 20 – Eclipse CDT “Refactor” operation

- Note that the usage of Eclipse “**Refactor**” allows to change according to “C” language characteristics (Functions, constant ...). This INGEDEV assistant will proceed in using this mechanism, for a great part of the renaming operation.
- Warning/error may also refer to elements to rename as “**constant**”, “**define**” “**struct name**”, Perform replacement according to “**MigrateToTPlus.xml**”.
- After some iterations or only one, if you have established a list of all the things to rename first, the TELIUM + framework project builds without errors.
 - Note that, as in TELIUM+ framework conflict with standard ‘C’ library have been removed “**implicit declaration of function**” from the ‘C’ library must be solved (= old TELIUM C library like function not renamed) because not error will be detected at “**link**” phase.
- At this step the project can be considered as migrated to TELIUM+ framework

5.3 Migration to TELIUM + framework specific cases

5.3.1 CPP project migration to TELIUM + framework

‘CPP’ usage is improved:

- The “**Enable C++**” property does not exist anymore in TELIUM + project properties (activated by default)
- Only one definition for an API whatever the used language ‘C’ and “**CPP**”. See the example below:
 - TELIUM 2 framework:
 - ‘C’ => “fopen”
 - ‘CPP’ => “eft_fopen”
 - TELIUM + framework:
 - ‘C’ or ‘CPP’ => “Telium_Fopen”
- No more specific libraries for CPP. See the example below:
 - TELIUM 2 framework:
 - ‘C’ => “eft30.lib”
 - ‘CPP’ => “eft30cpp.lib”
 - TELIUM + framework:
 - ‘C’ or ‘CPP’ => “eft30tplus.lib”
- The constraints mentioned in INGEDEV help contents about using “**new /delete**” operators disappear in TELIUM + framework. See the focused text in green in following picture.
- The constraints mentioned in INGEDEV help contents about mixing calls to TELIUM APIs in both languages disappear in TELIUM + framework. See the focused text in blue in following picture.

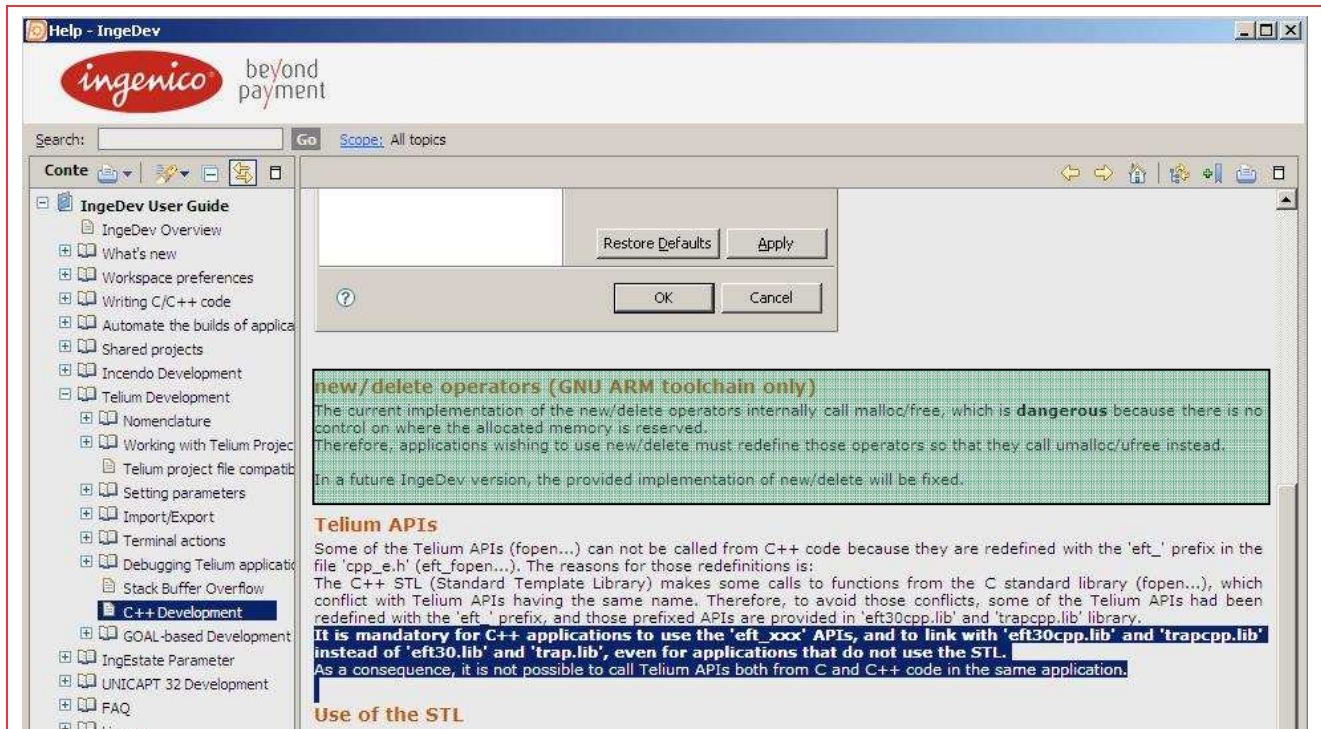


Figure 21 – CPP constraints removed with TELIUM + framework

5.3.2 TELIUM “vsprintf” specificity

Although is a pure logical function, « **vsprintf()** » is not currently provided by the standard C library of the used GCC tool chain to build the project: it’s delivered in SYSTEM TELIUM component and depends upon the tool chain used to build the SYSTEM which is not GNU GCC.

This specificity has created a mistake when GNU GCC was introduced to build the application: the parameter exchange in particular “**va_list**”, to work has to be passed as address “**&va_list**”.

A compliant code looks like the following line:


```
int MySprintf(char *s,const char *fmt,...)
{
    va_list    arglist;
    va_start(arglist, fmt);

    #if defined __GNUC__
        return(vsprintf(s, fmt, &arglist));
    #else
        return(vsprintf(s, fmt, arglist));
    #endif
} /* end MySprintf */
```

The major consequence is that the written lines are not more compliant with ‘C’ standard.

To remove this conflict with the standard C library, in the TELIUM + framework, this function is considered as specific TELIUM API. It is renamed in “**Telium_Vsprintf**”: the INGEDEV assistant will take in charge this renaming operation.

To use the standard C library function, the programmer must treat it manually:

| | | |
|---|---|---------------|
|  | TELUM2 to TELUM+ framework migration -HOW TO | ICO-OPE-00802 |
|---|---|---------------|

- Rename “**Telium_Vsprintf**” in “**vprintf**”
- Change “**&va_list**” parameter to return to C standard.

Note: When using C standard function, the size of application will be increased of more than 20 Kbytes.

5.3.3 TELIUM “RegisterPowerFailure” deprecation specific case

«**RegisterPowerFailure()**» is deprecated but during migration step inside terminal, from the application built with T2 framework to a next release built with T+ framework, its call is needed **one time** to recover the saved data and to transfer them to a new storage media.

So a new function has been created in TELIUM+ framework to fit this requirement (and only this requirement): «**Telium_RegisterPowerFailure()**». This function is only available in TELIUM 2 hardware context in which the feature was existing.

When you will replace this backup feature by a FILE system solution, take care to:

- Verify in each new version of your application, that this migration step of the data has been already passed.
- After data recovery and storage, do not forget to erase the information stored by the old mechanism.
- The new mechanism that you substitute to the old one must guaranty that file writing is only done when data really change.

5.3.4 Applications targeting all the terminals of TELIUM range

Some applications today have a very large usage perimeter in the TELIUM terminals range: they have been written by their authors to be supported on:


- Old machines with limited memory foot print and on which the evolution of SDK is never done or cannot be done.
- Recent machines on which the last SDK must be installed to satisfy applications new needs.

A priori they cannot be considered as eligible to TELIUM + framework.

However if you consider not the differences but the common things between T2 and T+ framework, it possible to create:

- 1 T2 framework project for old machines
- 1 T+ framework project for recent machines
- 1 “**Source container project**” contenting the common part and used by the 2 main projects.

In not duplicating a great part of the source code files, the cost of the application is lightly increased but the migration to T+ framework is possible for certain part of the machines in field that may continue to evolve .

| | | |
|---|---|---------------|
|  | TELIUM2 to TELIUM+ framework migration -HOW TO | ICO-OPE-00802 |
|---|---|---------------|

This Document is Copyright © 2009 by INGENICO Group. INGENICO retains full copyright ownership, rights and protection in all material contained in this document. The recipient can receive this document on the condition that he will keep the document confidential and will not use its contents in any form or by any means, except as agree beforehand, without the prior written permission of INGENICO. Moreover, nobody is authorized to place this document at the disposal of any third party without the prior written permission of INGENICO. If such permission is granted, it will be subject to the condition that the recipient ensures that any other recipient of this document, or information contained therein, is held responsible to INGENICO for the confidentiality of that information.

Care has been taken to ensure that the content of this document is as accurate as possible. INGENICO however declines any responsibility for inaccurate, incomplete or outdated information. The contents of this document may change from time to time without prior notice, and do not create, specify, modify or replace any new or prior contractual obligations agreed upon in writing between INGENICO and the user.

INGENICO is not responsible for any use of this device, which would be non consistent with the present document.

All trademarks used in this document remain the property of their rightful owners.