Transforming Automated Procedure Development with a state of the art IDE

Workshop on Simulation and EGSE for Space Programmes (SESP)
28-30 March 2017

ESA-ESTEC, Noordwijk, The Netherlands

Bjoern Kircher[1], Colin Borrett[2]

[1] Airbus DS GmbH
Claude-Dornier-Str.
88090 Immenstaad
Germany
Email: bjoern.kircher@airbus.com

[2] Airbus DS GmbH
Claude-Dornier-Str.
88090 Immenstaad
Germany
Email: colin.borrett@airbus.com

ABSTRACT

Developing automated monitor and control procedures for AIT, on-board operations or flight operations requires detailed information on how to operate the spacecraft with respect to the latest TM/TC definitions. In the case of AIT or functional verification activities, the procedure developer also requires detailed information about the spacecraft design, interfaces and the planned verification activities.
Existing procedure development environments offer little or no integrated data continuity from the Spacecraft Reference DB (SRDB) or the Functional Verification Management tool (FVM) to the procedure being developed. Most importantly, any evolution of data in the SRDB (e.g. update of TC parameter) or FVM (e.g. update of Test Specification) needs to be identified and updated manually by the developer, introducing the possibility of error or misconfiguration.
Therefore, in close cooperation with the end users, Airbus DS developed an integrated procedure development environment to ease and automate the process of data exchange/continuity, and provide a coherent snapshot of the data relevant to the developer. This environment further supports the full engineering process from using the actual design data (e.g. TM/TC), link to verification requirements and related test specification, up to a target language specific export towards the CCS or MCS.

The paper will introduce and detail the AIRBUS DS development, and the advantages that have been realized by using the integrated development environment for automated procedures.

CONTEXT

When discussing the development of automated procedures, the context described in this paper addresses mainly the writing of test procedures for Assembly Integration and Test (AIT) and Functional Verification (FV). However, we can consider that these are similar to those used to operate the satellite from ground and to procedures that are running on-board. Indeed, the basic principles of automated procedures are to send telecommands (TC), receive telemetry (TM) and use control structures to manage the test, and these principles are common to both test and operations procedures.

In order to effectively develop these procedures, very detailed knowledge about the system under control is needed, specifically regarding the control and monitoring of the spacecraft and the equipment on board. This knowledge begins with the overview of the spacecraft design, a detailed understanding of the TM/TC interface with all the possible commands and available monitoring parameters and packets, as well as the information about the test or operational need itself.
The management of the relevant design and system data is not a problem today and a tool landscape to manage and access the data is already well-established and integrated (see Fig. 1. Current tool landscape).

The two most important applications in terms of data management are the System Reference Data Base (SRDB) and the Functional Verification Manager (FVM).

- SRDB: The SRDB is the main tool based on the Airbus DS RangeDB infrastructure (RangeDB: The corner stone of the Model-Based Engineering approach at Airbus DS to manage data shared across engineering disciplines in a consistent manner). It is the central data set and repository containing the definitions and operational data utilized during the spacecraft engineering and development phases, as well as for both ground and flight data during spacecraft operations. It includes data such as product tree, equipment parameters, configuration data, TM/TC definitions, calibration curves and many more.
    - The SRDB is currently in use by 20 spacecraft (satellite & launcher) projects with more than 300 active users.
- FVM: Based on existing tools with over 10 years of heritage, and re-hosted using the flexible RangeDB infrastructure, FVM provides a modern digital continuity tool to support the FV process through :
    - Definition of test documentation data such as verification tasks, specification, procedure or report
    - Requirements traceability, from import of DOORS requirements, to the definition of the Test Specification and Test Procedure, to execution and storage of Test Sessions, and finally to generation of the Verification Control Document.

The SRDB is providing all the TM/TC definitions as well as the product structure of the satellite. The product structure and related data are used by FVM, together with a mapping to the detailed system requirements coming from DOORS (Rational Dynamic Object Oriented Requirements System – a requirement management tool). Thanks to the system design data from SRDB and the requirements from DOORS, FVM is used to define the test specifications on high level. From these specifications, the test engineer can derive directly the test procedure to be executed and provide a link back to the tested requirement. It is important to note that test procedures in FVM can be manual procedures or automated procedures.
Both SRDB and FVM integrate an interface to the PLM system (Windchill) which allows an easy and direct access to documentation linked to SRDB or FVM data.

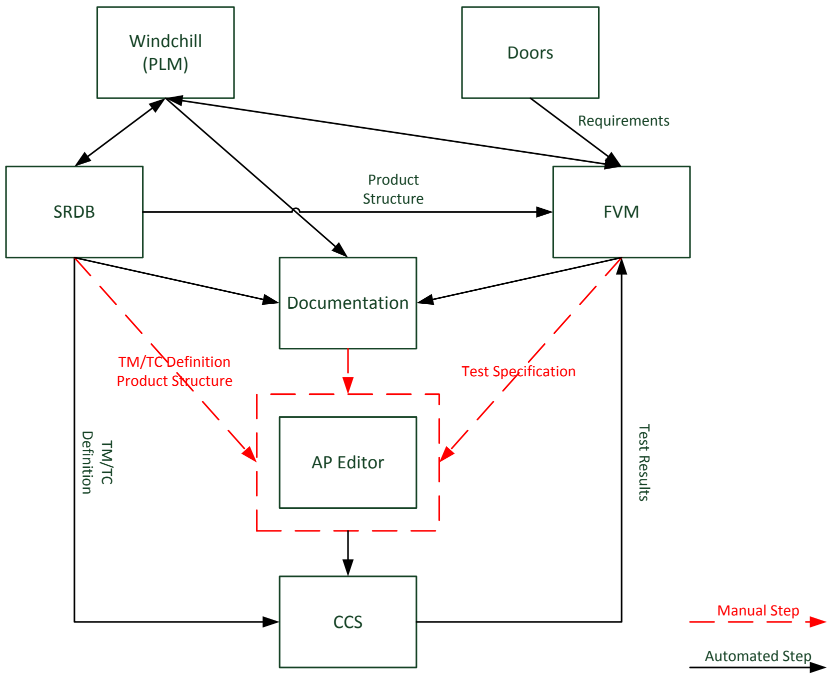


Fig. 1. Current tool landscape

The problematic point today is that even if the connection between SRDB and FVM is highly integrated, the tool to develop automated procedures that is using data from both is not integrated. It is still a more or less manual task to get the information regarding TM/TC, spacecraft design and test needs based on already digitally generated documents. Evolution of data in the SRDB (e.g. update of TC parameter) or FVM (e.g. update of Test Specification) needs to be identified and updated manually by the developer, introducing the possibility of error or misconfiguration.

To eliminate this source of problems the goal was to define a state of the art Procedures Development Environment (PDE) that is fully integrated in the already available tool landscape. This new application should allow:

- Usage of coherent data baseline release (from SRDB and FVM)
    - All tools, developers, testers are using the same data baseline
- Full digitalized process from design data (e.g. TM/TC), link to verification requirements and related test specification, up to the automated procedure running on CCS/MCS
- Automated generation of AP documentation and generation of over AP manual
- Common tool used by all users and for all use cases (AIT, FVI, Operations)
- State of the art IDE supporting features like:
    - Auto completion
        - Related to procedure language
        - Related to items in the spacecraft product structure
    - Tool tips
    - Syntax highlighting
    - Online help

DATA CONTINUITY

As identified previously, the primary goal of the PDE is to provide the means to connect the various disparate tools to be used without needing a dedicated export/import process, to synchronize release cycles so that all data being utilized is from the same baseline, and particularly to easily identify discrepancies in this baseline during the daily work flow of the procedure developer. The relationship of the AP editor to each of the tools is described in detail below.

Spacecraft Reference Database (SRDB)

In the existing tool landscape, when the SRDB architect releases a new database, it requires that the new version be installed or used on many systems (for example, the checkout system, or the data retrieval system), as shown in Fig. 2. Export / Release Cycle for SRDB data. For the test engineer implementing a procedure, this might mean that they would need to search the export by hand (using knowledge of the existing SCOS-2000 MIB tables), through the user documentation delivered with the SRDB (the TM/TC Handbook), or online using the CCS. It is often the case that instead of a simple description (e.g. Star Tracker Connection Test), the command or telemetry mnemonic is needed by the engineer, and the use of this also makes the resulting automated procedure less readable.

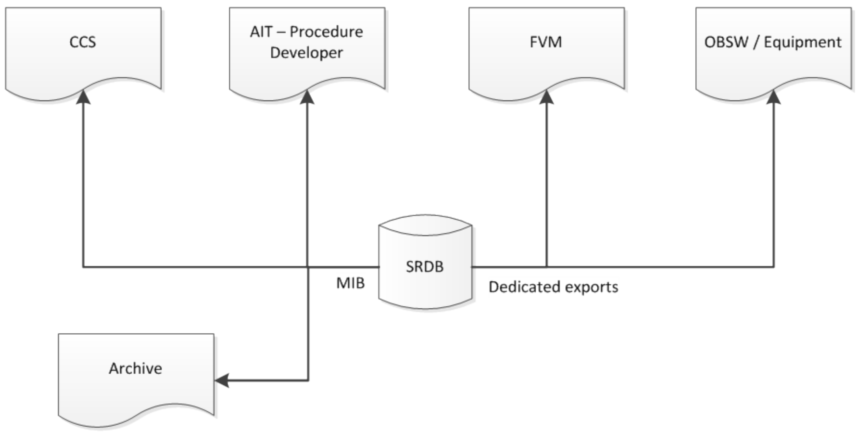


Fig. 2. Export / Release Cycle for SRDB data

Within the PDE, the goal of the RangeDB data continuity component was to remove the need for a new export/installation process, and to give the developer a direct view to the available TM/TC including the description, mnemonic, and parameters. This view is currently based on the product tree provided by RangeDB, but can be filtered or re-ordered depending on the developer preferences.

In a perfect world, the latest released database would be in use on all the dependent systems. However, it is clear that in the real world, the most recent database version is not always the same version that is installed on the bench, either because of the needs of the test operators, or an older version of OBSW which is installed. This can have a major impact when the test procedure is written, run, or debugged, leading to errors which need to be solved manually at the time of execution by the engineer, or which have to be justified in a post evaluation check of the results.

Instead, with the direct connection to SRDB, the test procedure engineer is able to control which version of the database is used for writing the procedure, and together with the syntax highlighter, provides feedback to the developer if their procedure is compatible with the version of the database selected (a similar function can be found in CCS5, where a procedure can be checked prior to execution). For example if the user has selected a version of the database which does not contain the service 17 command, then any procedure loaded will clearly identify the use of the command as an error.

With this component, the engineer writing the procedure knows that not only do they have the most recent or most applicable version for their needs, but that its contents are easily searchable and contain as much supporting information as possible.

Functional Verification Manager (FVM)

Within FVM, manual procedures (otherwise known as “step-by-step” or “operator” procedures) are maintained with the detailed steps which the operator may need to perform, alongside calls to automated procedures which are subsequently developed in the PDE. These “master” automated procedures provide the complete link to the test specification and the requirements to be verified.

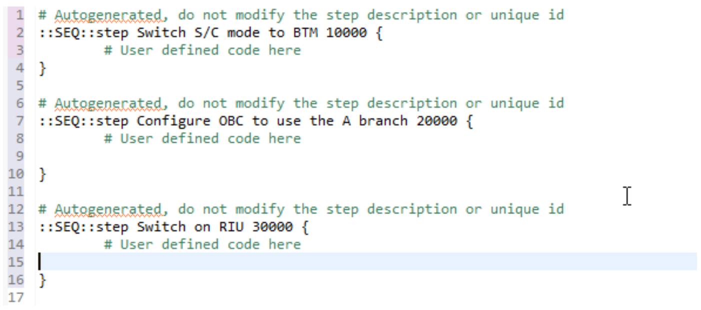
In order to prepare a master AP, the concept of a “skeleton” automated procedure is introduced, defining the main test steps which are to be executed. This provides a clear link between test specification steps (where the requirement is mapped) and the final test procedure execution (for requirement closeout). Specifically during post-processing, a unique identifier specific to each test step can be mapped to the execution results, and allow a customer to navigate directly to the location in the logs where their requirement has been declared as verified.

Take as an example the simple high level test specification steps as defined below:

- 10000 Switch S/C mode to BTM
- 20000 Configure OBC to use the A branch
- 30000 Switch on RIU

Note: Both FVM and the PDE are designed to allow nesting of test steps, and hence this simple example can be expanded to include sub-steps or other keywords as needed.

Through the connection with FVM and the test specification, the PDE can then generate the native language skeleton AP, which must be populated by the test engineer (for example by calling the required utility code to switch the spacecraft into Basic Test Mode). The example is shown in Fig. 3. Auto-generated skeleton.



```
# Autogenerated, do not modify the step description or unique id
::SEQ::step Switch S/C mode to BTM 10000 {
        # User defined code here
}

# Autogenerated, do not modify the step description or unique id
::SEQ::step Configure OBC to use the A branch 20000 {
        # User defined code here

}

# Autogenerated, do not modify the step description or unique id
::SEQ::step Switch on RIU 30000 {
        # User defined code here

}

```

Fig. 3. Auto-generated skeleton

We can take this concept one step further, and consider the impact when a test specification is up-issued to take into account a new requirement or review item. Whilst it is clear to the specification writer that the change may necessitate an update to the procedure, it will not necessarily be identified directly by the procedure developer.

From this, two scenarios can be imagined – in the first, an update to the test specification requires a completely new test step to be executed, or that a step is removed. In this case, the PDE identifies that the skeleton needs to be re-generated but with the mandatory option of retaining any code entered by the user. In the second scenario, an update to the test specification results in a change in the text (for example in the description of the test step, or in the criteria for test success). In this second case, the PDE identifies the modified steps, and shows the test procedure developer what data has been modified in the test specification.

Procedure Development

Part of the problem is the editor itself in which the automated procedures are developed. As presented earlier, the "AP Editor" is usually an integrated part of the CCS, or alternatively a plain text editor (particularly in the case where the CCS is an external product with few licenses available for developers). Both of these types of editor involve manual steps from the AP developer to write their procedures, and provide little or no information about the surrounding infrastructure or supporting documentation.

With projects becoming more generic, and to enable efficient reuse of already existing code, the procedure developer must have a view on already developed code, such as defined support libraries (which could be generic or project specific) or APs developed within the frame of current or previous projects. Particularly within the same project, a problem may have already been solved by another developer (for example the switch on sequence of particular equipment), and hence re-use of the existing code decreases development time and increases confidence in the final AP. With this in mind, it was identified that there was a clear need in the PDE of browsing the latest or historical versions of the APs in an online system or a configuration control system such as SVN or Git, and from here being able to easily retrieve or view reference source code.

As well as access to re-used code, when developing (or indeed executing) an automated procedure, it is important that the test executor has easy access to the reference documentation, such as safety precautions or operator user manuals, which might be applicable to the item / system that is under test. To this end, the Windchill Document Management system used by Airbus allows to generate external HTTP links to documents. These can be combined with the Product Tree so that the test procedure developer can see which documents are available for any particular item under test, and integrating this information into the PDE provides the developer with an easy overview of the documents which might be useful to them.

Test Execution and Debugging

Once the test procedure has been developed to a level which can be executed, the usual process is to move from the AP Editor and debug the procedure directly on the CCS connected to a simulator. This switch between environments can require a switch to a different machine (in the normal case that the CCS is shared between many developers), copying the files or uploading them to a configuration control system and updating the environment on the target machine, and then starting up the system into the configuration required to execute the test. Doing this many times for debugging a procedure increases the number of manual steps where errors can occur, which can take useful time away from the test procedure developer, and increase the unavailability of the test system to other developers.

Part of the identified solution to this was the development of a status webpage for each remote CCS machine, showing the current operator and identifying which systems are available. However, the ideal solution was to allow the developer to write, debug and finally execute the test procedure all in the same environment.

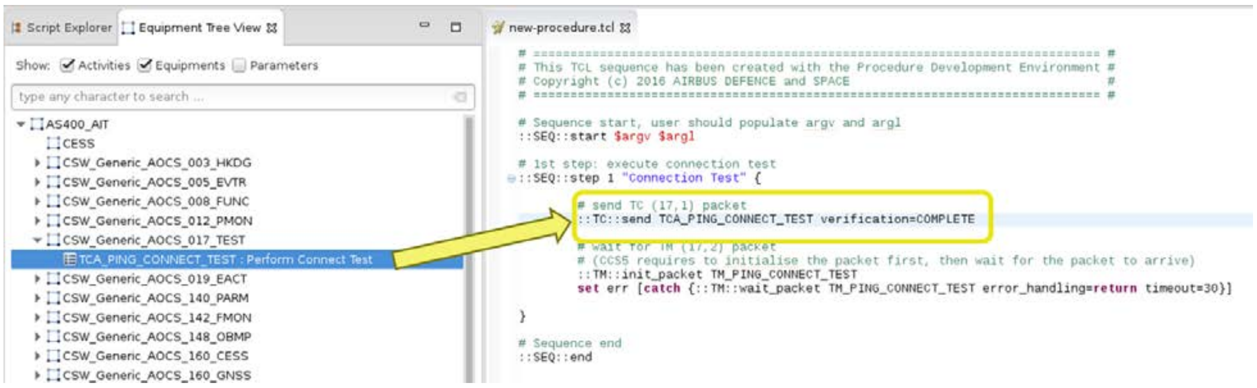
PDE – A STATE OF THE ART IDE

In addition to previously explained embedding of the PDE in the overall tool landscape in terms of processes and data continuity, it was also a goal to base the development environment on actual state of the art technologies. For usability, the PDE should include modern development features as they are known from classical SW development IDEs.

Specifically, standard features like code completion, syntax highlighting, error checking or refactoring capabilities should be available, as well as source navigators or navigation to and from references. However, the main focus was placed on support functions, enabling the tool to be intuitive and usable by non-SW developers. These support functions included features like tool tips that are displayed by hovering over a keyword (e.g. showing the description of a function), or simple navigation between e.g. main procedure and referenced sub procedures.

All these features primarily support the writing of the procedure itself, but it was also mandatory to implement and link the data coming from SRDB and FVM, and include those somehow in the developers view. Hence, similar to the features supporting the code writing of the target language, the same features should be available for the usage of data related to the spacecraft. An example of this is the auto completion of source code syntax, where the code completion should also work for data items like TMs or TCs. And further expanding on this idea, the auto completion should also work on the product structure itself. In this way, it should be possible for the user to get TM/TC data thanks to the location in the product tree (e.g. spacecraft -> subsystem -> equipment -> commands -> "on").

For usability of the tool, the access to TM/TC should also allow the now standard "drag and drop" features. Similar to a project navigator showing all the source files, this involves the use of a TM/TC tree based on the product structure of the spacecraft, where the user can navigate and simply drag and drop the related TM/TC to the procedure editor (see Fig. 4. TM/TC Tree View).


Script Explorer
Equipment Tree View
Show: Activities Equipments Parameters
type any character to search ...
AS400_AIT
CESS
CSW_Generic_AOCS_003_HKDG
CSW_Generic_AOCS_005_EVTR
CSW_Generic_AOCS_008_FUNC
CSW_Generic_AOCS_012_PMON
CSW_Generic_AOCS_017_TEST
TCA_PING_CONNECT_TEST : Perform Connect Test
CSW_Generic_AOCS_019_EACT
CSW_Generic_AOCS_140_PARM
CSW_Generic_AOCS_142_FMON
CSW_Generic_AOCS_148_OBMP
CSW_Generic_AOCS_160_CESS
CSW_Generic_AOCS_160_GNSS
new-procedure.tcl

```
# ============================================================================ #
# This TCL sequence has been created with the Procedure Development Environment #
# Copyright (c) 2016 AIRBUS DEFENCE and SPACE                                  #
# ============================================================================ #

# Sequence start, user should populate argv and argl
::SEQ::start $argv $argl

# 1st step: execute connection test
::SEQ::step 1 "Connection Test" {

    # send TC (17,1) packet
    ::TC::send TCA_PING_CONNECT_TEST verification=COMPLETE
    # wait for TM (17,2) packet
    # (CCS5 requires to initialise the packet first, then wait for the packet to arrive)
    ::TM::init_packet TM_PING_CONNECT_TEST
    set err [catch {::TM::wait_packet TM_PING_CONNECT_TEST error_handling=return timeout=30}]
}

# Sequence end
::SEQ::end
```

Fig. 4. TM/TC Tree View and Editor

Furthermore, in this TMTC view tooltips are implemented to show additional information such as descriptions coming from the SRDB or FVM, specifically when the user is pointing to one of the TMs or TCs. As mentioned previously, all the description of the test procedure and the related test steps which are managed in FVM are also included in the procedure source code.

Target Language Support

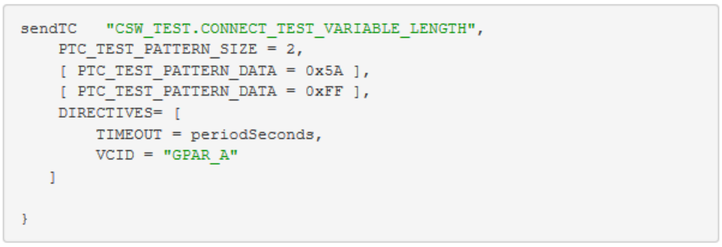
The target language supported by the PDE is the language that is used as procedure language in the CCS or MCS. The default language is currently TCL since this is the baseline for the CCS systems that are currently used with the PDE (see right hand part of Fig. **4**. TM/TC Tree View). But the PDE itself is not bound to one language, and thanks to a flexible plugin based concept, the PDE can be configured to support several target languages. It is even possible to develop procedures for different target platforms within one PDE instance. Hence, in addition to TCL there is already a prototype available to support Java APPG, the procedure language for the upcoming new checkout system EGS-CC.

Developing automated procedures using the target language of the CCS is mainly a request from the teams preparing the AIT and FV procedures. Furthermore, the possible use of a higher level language like a DSL (Domain Specific Language) was explicitly rejected since the procedures, especially for AIT, are often very complex with sophisticated features such as patching data from file or doing tricky mathematical computations.

Compared to the complex AIT and FV procedures the scope of procedures used for flight- or onboard operations are more basic and the use of a specific tailored language for that is convenient.

Domain-specific language

As explained before there are needs to use the target language of CCS/MCS directly, but especially for flight- or onboard operations it make sense to use a more simplified language dedicated to its use. Therefore, a so called DSL is also part of the PDE development, allowing the user to develop the procedures on a generic level with a subsequent transformation of the DSL procedure content to the target language. Thanks to this DSL, it is possible to generate procedure source code for different target platforms (e.g. if CCS and MCS are not based on the same system) based on a generic DSL based procedure.



```
sendTC   "CSW_TEST.CONNECT_TEST_VARIABLE_LENGTH",
    PTC_TEST_PATTERN_SIZE = 2,
    [ PTC_TEST_PATTERN_DATA = 0x5A ],
    [ PTC_TEST_PATTERN_DATA = 0xFF ],
    DIRECTIVES= [
        TIMEOUT = periodSeconds,
        VCID = "GPAR_A"
   ]

}
```

Fig. 5. DSL

Technologies used

The Procedure Development Environment is implemented with Java 8 as an Eclipse Rich Client Application (RCP) to provide a standalone, ready to use application. Eclipse is enriched with several plugins like the dynamic language toolkit for tcl to enable the use of tcl, the groovy language pack to support the DSL editor and subversive for svn integration.
In addition to the several eclipse plugins the Airbus RangeDB infrastructure is heavily used to provide features like HTML exports or exports to MS Works and Excel but also features like the code generation framework of RangeDB is used to transform the DSL to the target language.

CONCLUSION

The introduction of a harmonized Procedure Development Environment (PDE) has shown that an integrated approach using a state of the art IDE for automated procedures provides a lot of advantages, not only during the development, but also during maintenance phases. Closing the gap in the dataflow from system design and TM/TC data towards PDE is one of the most beneficial improvements achieved with the PDE. Thanks to a close link to the SRDB all the TM/TC definitions are available within the development tools, but even more crucial are the update and change processes from SRDB to PDE, where changes within the TM/TC definitions are directly propagated to the PDE and dependent procedures are highlighted. This is especially important in the early development phases where the design is not really stable but also in the later phases during maintenance where it is not simple to know all the impacts of data changes within the procedures.

The same is valid for data defined in FVM and used in PDE. All updates of test specifications are visible in the PDE and e.g. missing test steps in the automated procedures are highlighted and an actual status is generated showing the status of procedures that have to be developed (FVM baseline) versus the procedures that are already done.

In addition to a consistent and continuous use of data, the simplifications and supporting functions of the IDE contributes also to a large extend to the overall performance improvement in developing automated procedures. From a user perspective, contextual help like auto completion, navigation to reference based on procedure code (also based on system design data), has been one of the key assets.

In summary, the introduction of the PDE showed the envisaged advantages and process improvements. The next steps will be to link it closer to EGS-CC and further improve the user experience.