DADF Facility
US Detailed Design Specification - Workstation Software
## Document Change Record

<table>
<thead>
<tr>
<th>Issue/Revision</th>
<th>Date</th>
<th>DCN No.</th>
<th>Author</th>
<th>Changed Pages/Paragraphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>02.07.98</td>
<td></td>
<td>Sabine Nordmann</td>
<td>Initial Release</td>
</tr>
<tr>
<td>1.1</td>
<td></td>
<td></td>
<td></td>
<td>Add User Station detailed MMI design</td>
</tr>
<tr>
<td>2.0</td>
<td></td>
<td></td>
<td>Sabine Nordmann</td>
<td>Add VCSLog;Rev. 1.2 and VCSSUPER;Rev. 1.2 Reference Manuals</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Add User station MMIs design</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See change bars in sect. 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sec. 4 updated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sect. 5 updated to new requirements and section numbering</td>
</tr>
<tr>
<td>3.0</td>
<td>20.08.99</td>
<td></td>
<td>Sabine Nordmann</td>
<td>Updating Annex A: User station MMIs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See change bars in sect. 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sec 4. Completely updated</td>
</tr>
</tbody>
</table>
# Table of Contents

**DOCUMENT CHANGE RECORD** ................................................................................................... I

**TABLE OF CONTENTS** .............................................................................................................. II

**LIST OF FIGURES** ..................................................................................................................... V

**LIST OF TABLES** ........................................................................................................................ VI

3 **DETAILED DESIGN** .................................................................................................................. 3-1

3.1 **INTRODUCTION** ...................................................................................................................... 3-1

3.1.1 System Overview ................................................................................................................... 3-1

3.1.2 Background Processing ......................................................................................................... 3-1

3.1.3 Foreground Processing ......................................................................................................... 3-2

3.2 **DESIGN DECOMPOSITION** .................................................................................................... 3-3

3.2.1 Introduction .......................................................................................................................... 3-3

3.2.2 Services .............................................................................................................................. 3-3

3.2.2.1 US_Server .................................................................................................................... 3-3

3.2.2.2 VCSSUPER .................................................................................................................. 3-3

3.2.3 Processes ............................................................................................................................ 3-4

3.2.3.1 VCSLog ..................................................................................................................... 3-4

3.2.4 Threads ................................................................................................................................ 3-4

3.2.4.1 File Assembler .......................................................................................................... 3-4

3.2.4.2 AdminOnlineBuffer ................................................................................................. 3-5

3.2.4.3 PassXRITDataToDise ............................................................................................... 3-7

3.2.4.4 Manual decryption ....................................................................................................... 3-7

3.2.4.5 Manual decompression ............................................................................................... 3-7

3.2.5 MMI call backs ..................................................................................................................... 3-7

3.2.5.1 Configuration Control ................................................................................................. 3-7

3.2.5.2 Quicklook of historical data ....................................................................................... 3-12

3.2.5.3 Quicklook of realtime data ......................................................................................... 3-12

3.2.5.4 Data stream .................................................................................................................. 3-12

3.2.5.5 SKU access ................................................................................................................... 3-12

3.2.6 Libraries ............................................................................................................................... 3-12

3.2.6.1 MUBM and SKU (HW implementation) ..................................................................... 3-12

3.2.6.2 Log interface .............................................................................................................. 3-12

3.2.7 Data Stores .......................................................................................................................... 3-13

3.2.7.1 Online Storage ............................................................................................................ 3-13

3.2.7.2 Trace Buffer ............................................................................................................... 3-13

3.2.7.3 Log file and buffers .................................................................................................... 3-13

4 **SOFTWARE ITEMS DETAILED DESIGN SPECIFICATION** .................................................. 4-1

4.1 US_SERVER ............................................................................................................................... 4-1

4.1.1 US::Server .......................................................................................................................... 4-1

4.2 US_FILEASSEMBLER .................................................................................................................. 4-2

4.2.1 US::FileAssembler .............................................................................................................. 4-2

4.2.2 US::MUBM_Reception ...................................................................................................... 4-4

4.2.3 US::MUBM_VCDUTrace .................................................................................................... 4-7

4.2.4 US::MUBM_SPTRACE ....................................................................................................... 4-9

4.2.5 US::MUBM_Monitoring ..................................................................................................... 4-11

4.2.6 US::OCVCDU ..................................................................................................................... 4-13

4.2.7 US::OSourcePackets .......................................................................................................... 4-17

4.2.8 US::OTransportFile .......................................................................................................... 4-22

4.3 ADMINONLINESTORAGE ....................................................................................................... 4-26
7 ANNEX A: USER STATION MMIS ................................................................. 7-1

7.1 INTRODUCTION .................................................................................. 7-1

7.1.1 Purposes ......................................................................................... 7-1

7.1.2 MMI Tools and Environments ...................................................... 7-1

7.1.3 Story Boards .................................................................................. 7-1

7.1.4 Hierarchy diagram ........................................................................ 7-2

7.2 OVERVIEW ............................................................................................. 7-2

7.3 USER STATION BANNER ..................................................................... 7-4

7.4 DATA STREAM MONITOR SCREENS ................................................... 7-5

7.4.1 SKU Status ..................................................................................... 7-9

7.5 VIEWERS ............................................................................................... 7-10

7.5.1 Realtime Image Viewer ................................................................... 7-10

7.5.2 Historical Image Viewer ............................................................... 7-13

7.5.3 File viewer – ASCII&HEX ............................................................ 7-14

7.5.4 Trace Packet Viewer ....................................................................... 7-18

7.5.4.1 VCDU Trace Viewer ................................................................. 7-18

7.5.4.2 VCDU Trace Details ................................................................. 7-21

7.5.4.3 Source Packet Trace Viewer ..................................................... 7-22

7.6 CONTROL ............................................................................................... 7-24

7.6.1 User Station Configuration Control ........................................... 7-24

7.6.1.1 General tab ................................................................................ 7-26

7.6.1.2 Network tab .............................................................................. 7-28

7.6.1.3 Automatic Transfer .................................................................. 7-29

7.6.1.4 Buffer Sizes .............................................................................. 7-32

7.6.1.5 Base Band ................................................................................ 7-34

7.6.1.6 File Assembler .......................................................................... 7-36

7.6.1.7 Log ............................................................................................ 7-38

7.6.2 Station Key Unit Access ............................................................... 7-40

7.6.3 Edit Transfer Filter .......................................................................... 7-42

7.6.3.1 Edit Filter Entry ........................................................................ 7-46

7.7 LOG ....................................................................................................... 7-48

7.7.1 Realtime logview ............................................................................ 7-48

7.7.2 Historical logview ........................................................................ 7-51

7.8 FILE MANAGER ..................................................................................... 7-55
8 ANNEX B: VCS LOG

8.1 INTRODUCTION ............................................................................................................. 8-1
8.2 IMPLEMENTATION ............................................................................................................ 8-1
   8.2.1 Introduction ........................................................................................................... 8-1
   8.2.1.1 Process Synchronization ................................................................................ 8-1
   8.2.1.2 Memory Mapped Files .................................................................................... 8-1
   8.2.1.3 Dynamic Link Libraries .................................................................................. 8-2
   8.2.2 Programming the DLL ....................................................................................... 8-2
      8.2.2.1 Initialization ................................................................................................ 8-2
      8.2.2.2 Administrative Information ...................................................................... 8-4
      8.2.2.3 Implementing the Access to a Memory Mapped File .............................. 8-5
   8.2.3 Implementing the Server ....................................................................................... 8-5
      8.2.3.1 Description .................................................................................................. 8-6
      8.2.3.2 Writing Log Messages to Network Directories ........................................ 8-6
         8.2.3.2.1 Implementation ...................................................................................... 8-6
         8.2.3.2.2 Configuration ......................................................................................... 8-6
   8.3 INTERFACE DESCRIPTION ....................................................................................... 8-7
      8.3.1 writeLogMessage ............................................................................................ 8-7
      8.3.2 writeLogTimeMessage .................................................................................... 8-8
      8.3.3 GetLogLevel ..................................................................................................... 8-8
      8.3.4 IsLog ................................................................................................................ 8-9
   8.4 TOOLS ......................................................................................................................... 8-10
      8.4.1 CLEANUP ...................................................................................................... 8-10
      8.4.2 LOGVIEWER .................................................................................................. 8-11
      8.4.3 VCS_SUPER ................................................................................................. 8-12
      8.4.4 RLM_UI ........................................................................................................... 8-12

9 ANNEX C: VCSSUPER REFERENCE MANUAL .................................................................. 9-1

9.1 INTRODUCTION ............................................................................................................. 9-1
9.2 IMPLEMENTATION ......................................................................................................... 9-1
   9.2.1 Making Use of the Operating System ................................................................... 9-1
      9.2.1.1 Processes ................................................................................................... 9-1
         9.2.1.1.1 Introduction ............................................................................................. 9-1
         9.2.1.1.2 Starting Processes ................................................................................... 9-1
         9.2.1.1.3 Controlling ............................................................................................... 9-2
         9.2.1.1.4 Performance Management ................................................................. 9-2
      9.2.1.2 Services ....................................................................................................... 9-2
      9.2.2 Controlling VCSSUPER ................................................................................... 9-3
         9.2.2.1 Registry Database ..................................................................................... 9-3
            9.2.2.1.1 Logging on VCSSUPER ...................................................................... 9-3
            9.2.2.1.2 Logging off ........................................................................................... 9-5
      9.2.3 Implementing VCSSUPER ............................................................................... 9-5
List of Figures

Figure 3-1: System Overview ..................................................................................................................3-1
Figure 3-2: Background Hierarchy ........................................................................................................3-2
Figure 3-3: MMI hierarchy ......................................................................................................................3-3
Figure 3-4: FileAssembler Threads .......................................................................................................3-4
Figure 3-5: AdminOnlineStorage Threads ............................................................................................3-6
Figure 3-6: Online Buffer Structure .....................................................................................................3-13
List of Tables

Table 3-1: ConfigDB............................................................................................................ 3-12
Table 5-1: Design Traces....................................................................................................... 5-3
3 Detailed Design

3.1 Introduction

3.1.1 System Overview

The context of the MSG User Station software is shown in Figure 3-1: System Overview below. It is the same for both User Station types (HRUS and LRUS). The ‘US Workstation’ rectangle contains all User Station specific software components. The FileAssembler, the AdminOnlineBuffer and the SKU access-MMI are communicating with the MUBM via a SCSI-interface ([SCSI],[USDDS-B]) to receive and process the incoming data. The PassXRITDataToDise transfers the processed data to the DADF-DISE via network using TCP/IP. The ConfigDB contains all configuration parameters necessary to control the User Station functionality including parameters for the MUBM. Because the configuration handling for the User Station is very simple the NT registry is used for the ConfigDB. The logging mechanism is implemented by the VCSlog facility [see Annex B]. It is existing software at VCS. With small modifications it is used for the User Station event logging. The data connection to the OFL (drawn with broken lines) is no data stream realized in the User Station software. But it is possible to configure an automatic file transfer of the received and processed LRIT/HRIT files to any reachable location in the network as well to the DADF-OFL.

![Figure 3-1: System Overview](image)

The User Station software components can be divided in two major parts:

3.1.2 Background Processing

The automatic reception and data processing and the log facility are running in the background permanently while the Windows NT system is up and running. It is independent of a user is be logged in.
The background processing is implemented using Windows NT services. The executable US_Server and VCSSUPER (used for VCSLog) are installed as an service. Their subordinated processes of VCSSUPER and threads of US_Server are created by them. Figure 3-3: Background thread hierarchy gives an overview of the background components at the User Station.

Figure 3-1: Background Hierarchy

3.1.3 Foreground Processing
This includes the user interface (US_MMI screens) itself and the processing which fulfils the action requested by the user.

Figure 3-3: MMI hierarchy gives an overview about the functionality of the foreground processing. See Annex A for MMI prototyping.
3.2 Design decomposition

3.2.1 Introduction

The subsection of the design decomposition describes each main component shown in the diagrams before. Common components used for all software items are:

- Each software part creates log messages and writes them to a log file. This fact is not explicitly shown in the diagrams.
- MUBM and SKU interfaces are not described in detail in the diagrams. An extra section is added.

3.2.2 Services

3.2.2.1 US_Server

The US_Server is implemented as Windows NT service using the CNTService class. After start it sets up the configuration for this User Station software and creates and controls the subordinated threads. They are described in detail under section 3.2.3.

3.2.2.2 VCSSUPER

The VCSSUPER creates and controls the VCSLog process. It is an existing software package implemented by VCS. See [see Annex C] for detailed description.
3.2.3 Processes

3.2.3.1 VCSLog

VCSLog takes all log messages sent by software parts and puts them in a memory map, to store that log data very fast. The content of the memory map is then stored into log files. It is a VCS COTS software package. See [see Annex B] for a detailed description.

3.2.4 Threads

3.2.4.1 File Assembler

The major task of the File Assembler is to receive VCDUs and assemble – as last step – LRIT/HRIT files. If a LRIT/HRIT file is completed, it is passed to the ‘AdminOnlineStorageBuffer’ process thread for further processing. Routinely the status information created by the MUBM is read and prepared for the ‘Processing Status Monitor’. This is called the operational mode.

The File Assembler is also able to work in two different trace modes, which are selected by the user via the control MMI. These modes are implemented for test purposes.

If the File Assembler is switched to the VCDU trace mode, it writes each received VCDU extended by the reception timestamp and the attached quality information a result indicator of the Reed Solomon decoder to a – so-called – trace buffer. Further processing is disabled.

If the File Assembler is switched to the Source Packet trace mode, it assembles received VCDUs to Source Packets and writes these Source Packets extended by the reception timestamp, the CRC-result, the source packet length and a count, indicating how many filler bytes are inserted, to the trace buffer.

The lifetime of each VCDU respective Source Packet in the trace buffer is user defined but it is limited to maximal 15 minutes.

In each mode the VCDU attached quality information are processed and passed to the monitoring MMI.

The tasks of the FileAssembler are assigned to different software threads:

![Figure 3-1: FileAssembler Threads](image-url)
♦ MUBM_Monitoring

It reads in fixed intervals the monitoring data from the MUBM. The quality information passed by the MUBM are logged on monitoring level to the log file. If there are log messages available at the mubm, they will be read and – depending on their mubm specific log level- logged to the log file, and passed them as link status to the 'Datastream Monitor' MMI. The received MUBM Timer Value is assigned to the current system time, so that each MUBM Timer value is converted to a system time value.

♦ MUBM_Reception

This thread is active only in operational mode. It receives the VCDU data stream from the MUBM and assembles source packets from VCDU's and XRIT files from the source packets. It stores the contents of the received XRIT file into an object and passes that object with attached quality information to the next processing stage 'AdminOnlineBufferStorage'. The quality-information indicates the per cent of corrupted data in the received file and the reception time of the first VCDU assembled to this XRIT file.

♦ MUBM_VCDU_Trace

This thread is running only in VCDU or source packet trace mode. Similar to the MUBM_Reception thread, this one receives the VCDU data stream from the MUBM. In VCDU trace mode the VCDU packages – with attached quality and time information - are stored in the cyclic trace buffer.

♦ SP_Trace

This thread is running only in source packet trace mode. In source packet trace mode source packets are assembled and stored – also attached by quality and time information – in the cyclic trace buffer.

One trace buffer contains either VCDU trace information or source packet trace information, not both. If trace mode is switched on, the trace thread is started, opens the suitable trace buffer file, map it to the memory and store the trace data. If the trace mode is switched off, the contents of the trace buffer is stored to the file and the file is closed.

3.2.4.2 AdminOnlineStorage AdminOnlineBuffer

The AdminOnlineStorage_AdminOnlineBuffer controls the complete LRIT/HRIT file processing internal to the MSG User Station. It gets the received XRIT files from the FileAssembler, analyses its contents, defines a processing control record for the OnlineBufferControl_XRITProcessing, stores the XRIT file and queues the further data processing.

The processing control record contains information about the reception quality, storage location, further processing incl. Network transfer and the result of each processing stage. A time stamp is assigned to each record, which determines the deletion time for that file.

A short description of the processing sequence for one received LRIT/HRIT file, which needs all processing steps, is given below:

♦ Setup online buffer control record
♦ Store received file in online buffer
♦ Get pseudo noise key for decryption
♦ Perform decryption
♦ Perform decompression
☐ Transfer received and processed files to an offline region in the network, if configured
♦ Transfer monitoring data and the received and processed files to TransferToDISE, if configured
♦ Subsample image data and store/append to subsampled image file.
☐ Transfer monitoring data to TransferToDISE
♦ transfer received and processed files to an offline region in the network, if configured
♦ After storage duration, delete assigned files and control record

3-5
Each task described below is assigned to a software thread, which is shown in the chart below. Additionally an overview about data access is added to that chart.

Figure 3-1: AdminOnlineStorage Threads

- **AdminOnlineStorage**
  - **AdminOnlineBuffer**
    - Controls the activities of the processing threads.
  - **Decryption**
    - This thread passes pseudo noise key to the pattern generator to trigger start of new data. A new XRIT file is created in memory dedicated to the decryption processing. The decryption process is started. A set of pseudo noise pattern is retrieved from the generator and XORed with the encrypted data field of the XRIT file. Store the result in the decrypted data field. A completion status is stored in the online buffer control record and – depending on the state - the further processing is modified.
  - **FilteredTransfer**
    - If a LRIT/HRIT file after any processing step matches the conditions set by the active transfer filter, it is copied to a location, which is defined either by the transfer root parameter and a relative path assigned in the filter or the FTP parameter, if configured in the ConfigDB.
  - **StoreXRITFiles**
    - It gets completely received XRIT files from the FileAssembler and creates an online buffer control record to define the further processing. After completion of that administration work, the file is passed to its first processing stage.
  - **Decompression**
    - The decompression is started with the compressed data field and the resulting data are stored in the decompressed data field created by the decompression function.
Subsampling

Takes the data field of an image file, reduces the size and store the result in a suitable subsampled file. In case of a segmented image, each segment file is appended to the existing part.

ProvideAndRequestKeys

The SKU Control data are connected to this thread. The ProvideAndRequestKeys-thread provides EKM-files containing a set of public keys for the connected station key unit and on the other hand it requests the generation of a pseudo noise key for the decryption process.

ProvideNetData

Pass XRIT- monitoring data (XRIT file header and processing results) and -file data to the PassXRITDataToDise thread

OnlineBuffer

It consists of a data store for the file data and a buffer for the processing control data corresponding to the stored file data. The directory structure under the OnlineBuffer root is designed by the product-IDs defined for the annotation header and the file types. For each received XRIT file a control record is created. It is set up with reception information and controlling information for further data processing required for that file.

3.2.4.3 PassXRITDataToDise

This thread receives the MonitoringData and TransferFiles from the AdminOnlineBuffer.

The MonitoringData, containing quality information about the result of the reception, decryption and decompression processing, are passed to the DISE-Element, if the connection is established or stored in a buffer, if not. The content of the buffer is completely passed to the DISE-Element as soon as the link is created.

The TransferFiles are passed to the DISE-Element, if the corresponding link exist or ignored, if not. This thread is only available, if the MSG User Station is a DADF-component. A standalone version does not support data transfer to the network.

3.2.4.4 Manual decryption

After a user selection, the file manager starts the manual initiated decryption facility. It takes the selected XRIT file, request the pseudo-noise-key from the station key unit and starts the decryption. If the PBK storage of the needed key is updated after the file was received the US software is assuming, that no valid key is available for the decryption process. The user will be informed about this situation and has to confirm the continuation of the processing.

3.2.4.5 Manual decompression

After a user selection, the file manager starts the manual initiated decompression. It takes the selected XRIT file and starts the decompression.

3.2.5 MMI call backs

3.2.5.1 Configuration Control

The configuration-control check user input to the configuration and store them to the configuration control in the Windows NT registry. Each software item, which is controlled by the configuration during runtime, has to setup an event to its configuration parameter. Then the system informs the software item about the modifications. The contents of the configuration data base is described in the following:

Rel. Key: defines the key beginning at the registry root of the MSG User station configuration.
Name: define a value name defined under the specified key
Type: defines the type of the value
Description: describes the contents of that parameter.
<table>
<thead>
<tr>
<th>Rel. Key</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ConfigVersion</td>
<td>String</td>
<td>Version of this Configuration DB is set to 0.0</td>
</tr>
<tr>
<td>General</td>
<td></td>
<td></td>
<td>Common parameter</td>
</tr>
<tr>
<td></td>
<td>SoftwareRevision</td>
<td>String</td>
<td>Version of this MSG-User station software</td>
</tr>
<tr>
<td></td>
<td>DataRoot</td>
<td>String</td>
<td>Root location of the User station data</td>
</tr>
<tr>
<td>General/log</td>
<td></td>
<td></td>
<td>Configuration of the common log file</td>
</tr>
<tr>
<td></td>
<td>LogFileName</td>
<td>String</td>
<td>Name of the general log file. All log message not assigned to any facility are written here.</td>
</tr>
<tr>
<td></td>
<td>LogLevels</td>
<td>String</td>
<td>Define the loglevels</td>
</tr>
<tr>
<td>Log</td>
<td>LogLevelRange</td>
<td>String</td>
<td>Defines which log levels – provided by the VCSLog facility are supported by the User station</td>
</tr>
<tr>
<td>General</td>
<td></td>
<td></td>
<td>Defines LogFileName and Loglevel for the general log file</td>
</tr>
<tr>
<td>MUBMIF</td>
<td></td>
<td></td>
<td>Same as before for the mubm if log file</td>
</tr>
<tr>
<td>SKUCTL</td>
<td></td>
<td></td>
<td>Same as before for the SKU Control</td>
</tr>
<tr>
<td>XRITDT</td>
<td></td>
<td></td>
<td>Same as before for the XRIT Data Transfer unit</td>
</tr>
<tr>
<td>XRITFA</td>
<td></td>
<td></td>
<td>Same as before for the File Assembling unit</td>
</tr>
<tr>
<td>XRITFP</td>
<td></td>
<td></td>
<td>Same as before for the XRIT file processing</td>
</tr>
<tr>
<td>MUBM</td>
<td></td>
<td></td>
<td>MUBM characteristics</td>
</tr>
<tr>
<td></td>
<td>ProductID</td>
<td>String</td>
<td>Contains the MUBM type connected: MUBM.LKIT or MUBM.HKIT.</td>
</tr>
<tr>
<td></td>
<td>ProductRevision</td>
<td>String</td>
<td>Revision of the MUBM</td>
</tr>
<tr>
<td></td>
<td>SCSIbusID</td>
<td>DWORD</td>
<td>NT SCSI addressing: SCSI bus no.</td>
</tr>
<tr>
<td></td>
<td>SCSIDeviceName</td>
<td>String</td>
<td>NT SCSI addressing: Name of the SCSI port adapter</td>
</tr>
<tr>
<td></td>
<td>SCSITarget ID</td>
<td>DWORD</td>
<td>SCSI Initiator ID of the MUBM</td>
</tr>
<tr>
<td></td>
<td>SKUavailWait</td>
<td>DWORD</td>
<td>Timeout for an SKU available check after startup</td>
</tr>
<tr>
<td>MUBM/Baseband/Parameter</td>
<td></td>
<td></td>
<td>MUBM current settings of the control parameter</td>
</tr>
<tr>
<td></td>
<td>FSUnlockThreshold</td>
<td>DWORD</td>
<td>Frame synchronizer unlock threshold</td>
</tr>
<tr>
<td>Rel. Key</td>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>FSLockThreshold</td>
<td>DWORD</td>
<td>2\textsuperscript{nd} Frame synchronizer unlock threshold</td>
</tr>
<tr>
<td></td>
<td>FSMaxNoBitErrors</td>
<td>DWORD</td>
<td>Frame synchronizer max. no bit errors</td>
</tr>
<tr>
<td></td>
<td>FSMaxNoBitSlips</td>
<td>DWORD</td>
<td>Frame synchronizer max. no bit slips</td>
</tr>
<tr>
<td></td>
<td>FWPatternLength</td>
<td>DWORD</td>
<td>Frame synchronizer pattern length; fixed 32</td>
</tr>
<tr>
<td></td>
<td>MUBM/Baseband/Parameter/Default</td>
<td></td>
<td>MUBM default settings of the control parameter</td>
</tr>
<tr>
<td></td>
<td>FSUnlockThreshold</td>
<td>DWORD</td>
<td>Frame synchronizer unlock threshold</td>
</tr>
<tr>
<td></td>
<td>FSLockThreshold</td>
<td>DWORD</td>
<td>2\textsuperscript{nd} Frame synchronizer unlock threshold</td>
</tr>
<tr>
<td></td>
<td>FSMaxNoBitErrors</td>
<td>DWORD</td>
<td>Frame synchronizer max. no bit errors</td>
</tr>
<tr>
<td></td>
<td>FSMaxNoBitSlips</td>
<td>DWORD</td>
<td>Frame synchronizer max. no bit slips</td>
</tr>
<tr>
<td></td>
<td>FSWPatternLength</td>
<td>DWORD</td>
<td>Frame synchronizer pattern length</td>
</tr>
<tr>
<td></td>
<td>MUBM/Baseband/Threshold</td>
<td></td>
<td>Defines the valid ranges of MUBM monitoring data before a new entry in the link status MMI is generated</td>
</tr>
<tr>
<td></td>
<td>FSMarkerErrors</td>
<td>DWORD</td>
<td>Frame synchronizer Marker Errors</td>
</tr>
<tr>
<td></td>
<td>ISAGC</td>
<td>DWORD</td>
<td>Input Signal AGC</td>
</tr>
<tr>
<td></td>
<td>ISEbNo</td>
<td>DWORD</td>
<td>Input Signal Eb/N0</td>
</tr>
<tr>
<td></td>
<td>ISOffsetFrequency</td>
<td>DWORD</td>
<td>Input Signal Offset Frequency</td>
</tr>
<tr>
<td></td>
<td>ShortIntervalSecs</td>
<td>DWORD</td>
<td>Interval for short measurements</td>
</tr>
<tr>
<td></td>
<td>LongIntervalSecs</td>
<td>DWORD</td>
<td>Interval for long measurements</td>
</tr>
<tr>
<td></td>
<td>ViterbiDecoderBER</td>
<td>DWORD</td>
<td>Viterbi Decoder Bit errors</td>
</tr>
<tr>
<td></td>
<td>RSLongPFL</td>
<td>DWORD</td>
<td>Reed Solomon Decoder Long PFL</td>
</tr>
<tr>
<td></td>
<td>RSShortPFL</td>
<td>DWORD</td>
<td>Reed Solomon Decoder</td>
</tr>
<tr>
<td></td>
<td>VDLongBER</td>
<td>DWORD</td>
<td>Viterbi Decoder Long BER</td>
</tr>
<tr>
<td></td>
<td>VDShortBER</td>
<td>DWORD</td>
<td>Viterbi Decoder Short BER</td>
</tr>
<tr>
<td></td>
<td>MUBM/Log</td>
<td></td>
<td>Configuration of the MUBM logging</td>
</tr>
<tr>
<td></td>
<td>LogFileName</td>
<td>String</td>
<td>Name of the MUBM log file. To this log file all MUBM logs are written.</td>
</tr>
<tr>
<td></td>
<td>LogLevels</td>
<td>String</td>
<td>Define the loglevels</td>
</tr>
<tr>
<td></td>
<td>SKUControl</td>
<td></td>
<td>SKU access configuration</td>
</tr>
<tr>
<td></td>
<td>PBKStateLocation</td>
<td>String</td>
<td>Specification of the file containing the current state of the loaded PBKs</td>
</tr>
<tr>
<td></td>
<td>PBKStateMapName</td>
<td>String</td>
<td>Specification of the previous defined file after it is mapped to the memory</td>
</tr>
<tr>
<td>Rel. Key</td>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>SKUNumber</td>
<td>String</td>
<td>SKU number returned with the MUBM characteristics</td>
</tr>
<tr>
<td>SKUControl/Log</td>
<td></td>
<td></td>
<td>Configuration of the SKU access logging</td>
</tr>
<tr>
<td></td>
<td>LogFileName</td>
<td>String</td>
<td>Name of the SKU log file. To this log file all messages are written, which are occurred during the SKU access.</td>
</tr>
<tr>
<td></td>
<td>LogLevels</td>
<td>String</td>
<td>Define the loglevels</td>
</tr>
<tr>
<td>XRTIDataTransfer/ DISE</td>
<td></td>
<td></td>
<td>Configuration of the data transfer to the DISE-Element</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
<td>DWORD</td>
<td>Defines if DISE is available</td>
</tr>
<tr>
<td></td>
<td>FileIntervalClear</td>
<td>DWORD</td>
<td>Count defining the filter for clear XRT files</td>
</tr>
<tr>
<td></td>
<td>FileIntervalComp</td>
<td>DWORD</td>
<td>Count defining the filter for the compressed XRT files</td>
</tr>
<tr>
<td></td>
<td>FileIntervalEnc</td>
<td>DWORD</td>
<td>Count defining the filter for the encrypted XRT files</td>
</tr>
<tr>
<td></td>
<td>FileIntervalMax</td>
<td>DWORD</td>
<td>Upper limit for the filter count</td>
</tr>
<tr>
<td></td>
<td>FileIntervalMin</td>
<td>DWORD</td>
<td>Lower limit for the filter count</td>
</tr>
<tr>
<td></td>
<td>FilePortId</td>
<td>String</td>
<td>Port ID of the DISE-element for File data transfer. The US is waiting for a link established to this port.</td>
</tr>
<tr>
<td></td>
<td>MonBufferDisabled</td>
<td>DWORD</td>
<td>Defines, if monitoring data shall be buffered during the link is down to DISE</td>
</tr>
<tr>
<td></td>
<td>MonBufferDuration</td>
<td>DWORD</td>
<td>Defines the max. storage time for monitoring data, if MonBufferDisabled is false.</td>
</tr>
<tr>
<td></td>
<td>MonPortId</td>
<td>String</td>
<td>Port ID of the DISE-element for Monitoring data transfer. The US is waiting for a link established to this port.</td>
</tr>
<tr>
<td>XRTIDataTransfer/ Log</td>
<td></td>
<td></td>
<td>Configuration of the data transfer logging</td>
</tr>
<tr>
<td></td>
<td>LogFileName</td>
<td>String</td>
<td>Name of the XRT Data transfer log file.</td>
</tr>
<tr>
<td></td>
<td>LogLevels</td>
<td>String</td>
<td>Define the loglevels</td>
</tr>
<tr>
<td>XRTIDataTransfer/ OFFL</td>
<td></td>
<td></td>
<td>Configuration of the automated data transfer to any reachable network destination.</td>
</tr>
<tr>
<td></td>
<td>FilterDirectory</td>
<td>String</td>
<td>Specify the location of the available filters</td>
</tr>
<tr>
<td>Rel. Key</td>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>FTPHostName</td>
<td>String</td>
<td>FTP Transfer: Specify name of the remote computer.</td>
</tr>
<tr>
<td></td>
<td>FTPIPAddress</td>
<td>String</td>
<td>FTP Transfer: Specify IP address of the remote computer</td>
</tr>
<tr>
<td></td>
<td>FTPPassword</td>
<td>String</td>
<td>FTP Transfer: password of the remote user</td>
</tr>
<tr>
<td></td>
<td>FTPUser</td>
<td>String</td>
<td>FTP Transfer: user name at the remote computer.</td>
</tr>
<tr>
<td></td>
<td>TransferFilter</td>
<td>String</td>
<td>Specify the active filter</td>
</tr>
<tr>
<td></td>
<td>TransferRoot</td>
<td>String</td>
<td>Specify the root for the automated file transfer on the remote computer.</td>
</tr>
<tr>
<td>XRTIDataTransfer/</td>
<td>LogFileName</td>
<td>String</td>
<td>Name of the OFFL Data transfer log file.</td>
</tr>
<tr>
<td>OFFL/Log</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LogLevels</td>
<td>String</td>
<td>Define the loglevels</td>
</tr>
<tr>
<td>XRTIFileAssembling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRCCheck</td>
<td>DWORD</td>
<td>Switch CRC check of source packets on or off</td>
</tr>
<tr>
<td></td>
<td>MeasurementInterval</td>
<td>SHORT-Array</td>
<td>Defines the measurement interval for the monitoring data. The array contains min, max, default and current value of this parameter.</td>
</tr>
<tr>
<td></td>
<td>TraceBufferDir</td>
<td>String</td>
<td>Specifies the location for the Trace buffer</td>
</tr>
<tr>
<td></td>
<td>TraceBufferStorageTime</td>
<td>DWORD</td>
<td>Defines the storage time of a packet in the trace buffer</td>
</tr>
<tr>
<td></td>
<td>TraceModeReceptionMode</td>
<td>String</td>
<td>Select Trace mode: NOOPERATIONAL, VCDUTRACE and SPTRACE</td>
</tr>
<tr>
<td>XRTIFileAssembling/</td>
<td>LogFileName</td>
<td>String</td>
<td>Name of the File assembling log file.</td>
</tr>
<tr>
<td>Log</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LogLevels</td>
<td>String</td>
<td>Define the loglevels</td>
</tr>
<tr>
<td>XRTIFileProcessing/</td>
<td>LogFileName</td>
<td>String</td>
<td>Name of the XRTIFileProcessing log file.</td>
</tr>
<tr>
<td>Log</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LogLevels</td>
<td>String</td>
<td>Define the log levels</td>
</tr>
<tr>
<td>XRTIFileProcessing/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OnlineStorage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Configuration of the online storage</td>
</tr>
</tbody>
</table>
Table 3-1: ConfigDB

<table>
<thead>
<tr>
<th>Rel. Key</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRITStorageRoot</td>
<td>String</td>
<td>Root of online buffer</td>
<td></td>
</tr>
<tr>
<td>XRITStorageTime</td>
<td>DWORD</td>
<td>Defining a storage time for a received XRIT file and the processing results.</td>
<td></td>
</tr>
<tr>
<td>StorageTime</td>
<td>DWORD</td>
<td>Defining a storage time for the image files containing sub sampled data.</td>
<td></td>
</tr>
<tr>
<td>ColorLocation</td>
<td>String</td>
<td>Location of the color tables selectable in the Quicklook display</td>
<td></td>
</tr>
<tr>
<td>QuicklookMonitoring</td>
<td>Used to indicate the realtime image viewer the available quicklookfiles.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2.5.2 Quicklook of historical data
The user selected subsampled file is read and display in the Historical quicklook MMI.

3.2.5.3 Quicklook of realtime data
The quicklook data for the currently received images are stored in memory using the generic server during the reception process. This actual content is passed to a MMI, if requested by the user.

3.2.5.4 Data stream
Status information from different processing stages is collected. If the data stream monitor MMI is started, its display is updated if the next monitoring data are available.

3.2.5.5 SKU access
The SKU Control mapped to this MMI callback synchronizes the access to the station key unit.

3.2.6 Libraries

3.2.6.1 MUBM and SKU (HW implementation)
The MUBM and the SKU are connected to the User Station computer by a SCSI interface via the same device.
Each component has its own logical unit number and the access to their functionality is implemented by a set of SCSI commands.
The interface is defined in detail by the ICD MUBM::Workstation [USDDS-B].

3.2.6.2 Log interface
See [Annex B] for detailed description. It is a VCS COTS software package. No further design is provided.
3.2.7 Data Stores

3.2.7.1 Online Storage

The US_OnlineBuffer contains all XRIT files received and processed. They are stored until a configurable time is passed. An US_OnlineBufferControl data store is provided, to administrate and control the contents of the US_OnlineBuffer.

The US_OnlineBuffer is structured into subdirectories starting from a US_OnlineBuffer root position. The directory structure is defined by XRIT filetypes and their product-IDs set in the annotation header. The following figure shows the structure of the US_OnlineBuffer:

![Figure 3-1: Online Buffer Structure](image)

The US_OnlineBufferControl contains a record for each received XRIT file. It stores:

- The file location in the online buffer
- Quality information, indicating how much of a LRIT/HRIT file is correct received, decrypted and decompressed. It is passed through the DISE-Element by the monitoring data,
- Control information for the data processing, like PN-Key and so on.

The US_OnlineBufferControl is stored in a memory-mapped section, which is assigned to disk file.

3.2.7.2 Trace Buffer

The Trace Buffer is created only during VCDU - or Source packet trace mode. If the system is switched from the operational to a trace mode, the trace buffer is created as a memory mapped file. All available packages attached by time – and quality information are stored in the trace buffer. After trace mode is switched off, the trace buffer is saved to disk for further analysing purposes.

3.2.7.3 Log file and buffers

The structure of a logfile respectively the logbuffer is defined by VCSLog[Annex B]. It is a VCS COTS software package. No further design is provided.
4 Software Items Detailed Design Specification

4.1 US::Server

4.1.1 US::Server

#ifndef _CSU_H_
define_CSU_H_
**************************************************************************
**** DADF MODULE HEADER ***

TYPE: Service

PURPOSE: background processing must implemented as service

FUNCTION: controls the automatic background processing

INTERFACES: See ‘INTERFACES:’ in the module declaration below

RESOURCES: --

REFERENCES: --

PROCESSING: - get process control configuration parameter
- initial access to the MUBM
- for each of ...
- US::US_FileAssembler
- US::US_AdminOnlineStorage
- US::US_SKUControl
do start main thread and setup controlling parameter
if ( Network transfer enabled )
then
    start and control US::US_PassXritDataToDise
    main thread
end if
- control running processes

DATA: See ‘DATA:’ in the class header below.

LOGIC: --

**************************************************************************
END MODULE HEADER ***
**************************************************************************/


#include "CNTService.h"
#include "CUSLog.h"

namespace US
{
    class CSU : public Util::CNTService
    {
    public:
        // DATA:

        // INTERFACES:

        // Constructor for software unit
        CSU
        {
            // Arguments:
            const std::string& i_ServiceName // Name of the service
        };

        // Implementation of the User station
        void Run();

    private:
        std::string m_ServiceName; // Name of the service
    };

} // end namespace US
#endif

4.2 US_FileAssembler

4.2.1 US::FileAssembler

/*****************************************************************************/
/* Eumetsat Meteosat Second Generation */
/* Data Acquisition and Dissemination Facility: DADF */
/* This software is developed by VCS Nachrichtentechnik GmbH under contract */
/* VCS      : V97.103.1008 */
/* Eumetsat : EUM/CO/97/499/YB */
/* Copyright VCS 1998 */
/* Applicable Third-party Software Licence Information: */
/* *****************************************************************/
/* Configuration Control Details from Visual Source Safe */
/* */
/* $Archive: $ */
/* */
/* $History: $ */
/* */
/*
   * INTERFACE DECLARATION
   */

#ifndef _FILEASSEMBLER_H_
#define _FILEASSEMBLER_H_

/**
  **  DADF MODULE HEADER   
  **
  TYPE: Thread
  PURPOSE: Interfacing the MUBM to the User station PC
  FUNCTION: Control the different VCDU reception units.
  INTERFACES: See 'INTERFACES:' in the module declaration below
  RESOURCES: --
  REFERENCES: MSI
  
  PROCESSING: Read configuration parameter from the registry and setup
  config_event to signaling config changes

  create MUBM_Monitoring Thread

  depending on the value of the ReceptionMode-config parameter
  create one of the reception threads:
  - MUBM_Reception (operational mode)
  - MUBM_VCDUTrace (vcdu trace mode)
  - MUB_SPTrace (source packet trace mode)

  do until shutdown or fatal error
  {
    Wait for one of the following events
    - thread terminates
    - shutdown event
    - Config changed

    switch on signaled event:
      case THREAD_TERMINATION
        if thread terminates unexspected
          then
            handle error
        else
          depending on the value of the ReceptionMode
          create one of the reception threads:
            - MUBM_Reception (operational mode)

  */
- MUBM_VCDUTrace (vcdu trace mode)
- MUB_SPTrace  (source packet trace mode)

end else

case SHUTDOWN_EVENT
    signal shutdown to all threads
    terminate this thread

case CONFIG_CHANGED:
    if (ReceptionMode changed)
        then
            request shutdown of the currently active
            reception process

        }

log process shutdown
set termination status;

DATA:  --

LOGIC:  --

**** END MODULE HEADER  ***
**************************************************************************
#include "BaseNT.h"
namespace US
{
    //INTERFACES:

    //
    void US_FileAssembler
    // Description:  FileAssembler function started as thread from the CSU.Run()
    // Arguments:
    {
        // Parameter not needed for function, but for CThread
        int* i_notUsed,
        // Shutdown event
        const HANDLE i_ShutdownEvent
    );
    // Returns:  --
    // Exceptions:  --

} //end namespace US
#endif

4.2.2  US::MUBM_Reception
**************************************************************************
* Eumetsat Meteosat Second Generation
* Data Acquisition and Dissemination Facility: DADF
* This software is developed by VCS Nachrichtentechnik GmbH under contract
* VCS  : V97.103.1008
* Eumetsat : EUM/CO/97/499/YB
*
* Copyright VCS 1998
* 
* Applicable Third-party Software Licence Information:
* 
*----------------------------------------------------------------------
* Configuration Control Details from Visual Source Safe
* 
*$Archive: $
* 
*$History: $
* 
*----------------------------------------------------------------------
* 
**************************************************************************
*/

#ifndef _MUBM_RECEPTION_H_
define _MUBM_RECEPTION_H_
/**************************************************************************
****  DADF MODULE HEADER   ***
**************************************************************************

TYPE: THREAD

PURPOSE: Independently access to MUBM between monitoring - and vcdu data transfer.

FUNCTION: VCDU reception and assembling of LRIT/HRIT files

INTERFACES: Using Util::CThread for the thread creation requires the Interface defined below. See 'INTERFACES:' in the module declaration below

RESOURCES: miniport driver (SCSI-access)

REFERENCES: ICD MUBM::Workstation [USDDS-B Annex-A] MSI

PROCESSING:
Open MUBM device for data reception
queue reception of VCDU_DATA
create Vcdu, SourcePacket and TransportFile object

do until shutdown or fatal error
{
    wait for one of the following events
    - shutdown event
    - vcdu data available

    switch on event
    case LOCAL_SHUTDOWN
release queued MUBM-command
terminate thread
case VCDU_DATA
  for all received VcduDataPackets in VCDU_DATA
    Analyse the vcdu quality data
    if important quality changes,
    then pass them to monitoring mmi
    if ( vcdu data available)
    then
      if ( Vcdu::Filler ( &BERCount))
        then Add BERCount to vcduBerCount,
        increment vcduIdleCount
      if ( Vcdu::Ignore)
        then log event
      if ( Vcdu::M_PduValid )
        then
          for all Sourcepackets Segments in M_Pdu(VCDU_DATA packet)
            Assemble Source packets
            if ( Source packet complete)
              then
                Assemble TransportFile and
                queue a completed file for furhter
                processing in the AdminOnlineBuffer
          end switch
    end switch
log thread shutdown
set termination status

DATA:

LOGIC:    --

**** END MODULE HEADER   ***
***************************************************************************/
#include "BaseNT.h"
#include <string>
namespace US
{
  // DATA:

  // INTERFACES:

  // Assembling of VCDUs to LRIT/HRIT Files
  // --------------------------------------
  void MUBM_Reception(
    // Arguments:
      // first parameter not used
      void *i_unused,
      // shutdown event
      const HANDLE i_shutdownEvent
  );

  // Returns:  --
  //
  // Exceptions:  --
}
4.2.3 US::MIUBM_VCDUTrace

/*reamark:***************************************************************************/
/* Eumetsat Meteosat Second Generation                                         */
/* Data Acquisition and Dissemination Facility: DADF                          */
/* This software is developed by VCS Nachrichtentechnik GmbH under contract    */
/* VCS : V97.103.1008                                                          */
/* Eumetsat : EUM/CO/97/499/YB                                                */
/* Copyright VCS 1998                                                          */
/* Applicable Third-party Software Licence Information:                       */
/*-----------------------------------------------------------------------------*/
/* Configuration Control Details from Visual Source Safe                      */
/* $Archive: $                                                               */
/* $History: $                                                               */
/*-----------------------------------------------------------------------------*/
/****************************************************************************
*
 ifndef _MUBM_VCDUTRACE_H_
 define _MUBM_VCDUTRACE_H_
 /***********************************************************************************/
**** DADF MODULE HEADER ***

TYPE: THREAD

PURPOSE: Independently access to MUBM between monitoring - and received data transfer.

FUNCTION: Receive VCDUs and -depending on the Trace mode - assemble source packets. Store VCDUs or Source packets with attached timing and status information to a trace buffer.

INTERFACES: Using Util::CThread for the thread creation requires the Interface defined below. See 'INTERFACES:' in the module declaration below

RESOURCES: miniport driver (SCSI-access)

REFERENCES: ICD MUBM::Workstation [USDDS-B Annex-A]
    MSI

PROCESSING:
queue reception of VCDU_DATA
create Vcdu object
create a trace buffer object and initialize
for VCDU trace
do until shutdown or fatal error
{
  wait for one of the following events
  - local shutdown event
  - vcdu data available

  switch on event
  case LOCAL_SHUTDOWN
    release queued MUBM-command
    terminate thread
  case VCDU_DATA
    for all received VcduDataPackets in VCDU_DATA
      Analyse the vcdu quality data
      if important quality changes,
        then pass them to monitoring mmi
      if ( vcdu data available)
        then
          if ( Vcdu::Filler ( &BERCount))
            then Add BERCount to vcduBerCount,
                Increment vcduIdleCount
          if ( Vcdu::Ignore)
            then log event
            store vcdu and its quality data to the
            trace buffer
        end for
  end switch
  log thread shutdown
  set termination status

DATA:
LOGIC: --

**** END MODULE HEADER ****
**************************************************************************
#include "BaseNT.h"
#include <string>

namespace US
{
  // DATA:

  // INTERFACES:

  // Assembling of VCDUs to LRIT/HRIT Files
  // --------------------------------------
  void MUBM_VCDUTrace
  (
    // Arguments:

#ifndef _MUBM_SPTRACE_H_
#define _MUBM_SPTRACE_H_

/* **************************************************************************

** DADF MODULE HEADER ***

TYPE: THREAD

 PURPOSE: Independently access to MUBM between monitoring - and received data transfer.

 FUNCTION: Receive VCDUs, assemble to source packets and store them with attached timing and status information to a trace buffer.

 INTERFACES: Using Util::CThread for the thread creation requires the Interface defined below.

 RESOURCES: miniport driver (SCSI-access)

***************************************************************************/

#endif

4.2.4 US::MUBM_SPTRACE

// unused

void* i_UNUSED,
    // shutdown event
    const HANDLE i_shutdownEvent
);

// Returns: --
// Exceptions: --

*/

#ifndef _MUBM_SPTRACE_H_
#define _MUBM_SPTRACE_H_

/***************************************************************************/
REFERENCES: ICD MUBM::Workstation [USDDS-B Annex-A]  
MSI

PROCESSING:
queue reception of VCDU_DATA
create Vcdu and Source packet object
create a trace buffer object and initialize for Sourcepacket trace
do until shutdown or fatal error {
    wait for one of the following events
    - local shutdown event
    - vcdu data available
    switch on event
    case LOCAL_SHUTDOWN
        release queued MUBM-command
        terminate thread
    case VCDU_DATA
        for all received VcduDataPackets in VCDU_DATA
            Analyse the vcdu quality data
            if important quality changes,
                then pass them to monitoring mmi
            if ( vcdu data available)
                then
                    if ( Vcdu::Filler ( &BERCount))
                        then Add BERCount to vcduBerCount,
                            Increment vcduIdleCount
                    if ( Vcdu::Ignore)
                        then log event
                    if ( Vcdu::M_PduValid )
                        then
                            for all Sourcepackets Segments in M_Pdu(VCDU_DATA packet)
                                Assemble Source packets
                                if ( Source packet complete)
                                    then
                                        store its contents to the trace buffer
                            end for
                end if
        end switch
    log thread shutdown
    set termination status

DATA:

LOGIC: --

**** END MODULE HEADER  ***
**************************************************************************/
#include "BaseNT.h"
#include <string>

namespace US {


// DATA:
// INTERFACES:
// Assembling of VCDUs to LRIT/HRIT Files
// --------------------------------------
void MUBM_SPTrace
{
    // Arguments:
    // unused
    void* i_unused,
    // shutdown event
    const HANDLE i_shutdownEvent
};
// Returns: --
// Exceptions: --
} #endif

4.2.5 US::MUBM_Monitoring

/**************************************************************************
** Eumetsat Meteosat Second Generation
** Data Acquisition and Dissemination Facility: DADF
** This software is developed by VCS Nachrichtentechnik GmbH under contract
** VCS : V97.103.1008
** Eumetsat : EUM/CO/97/499/YB
** Copyright VCS 1998
** Applicable Third-party Software Licence Information:
** Configuration Control Details from Visual Source Safe
** $Archive: $
** $History: $
** *-----------------------------------------------------------------------
** Applicable Third-party Software Licence Information:
** *-----------------------------------------------------------------------
** $Archive: $
** $History: $
** *-----------------------------------------------------------------------
**
#define MUBM_MONITORING_H_
#define MUBM_MONITORING_H_
/*** DADF MODULE HEADER ***

TYPE: THREAD
PURPOSE: Independently access to MUBM between monitoring - and received data transfer.

FUNCTION: Reading MUBM monitoring data and pass them to a MMI-client - if connected -

INTERFACES: Using Util::CThread for the thread creation requires the Interface defined below. See 'INTERFACES:' in the module declaration below

RESOURCES: DADF-COMS miniport driver (SCSI-access)

REFERENCES: ICD MUBM::Workstation [USDDS-B Annex-A] MSI

PROCESSING:
Read configuration parameter from the registry and setup config_event to signaling config changes

do until shutdown or fatal error
{
    wait for one of the following events
    - time out
    - shutdown event
    - config change

    switch on event
    case TIME_OUT:
        Request MUBM_Monitoring data from the MUBM device and log them at monitoring level
        if the mubm has log messages available then
            read the log messages from the mubm
            put them - depending on their log level - in the workstation log message format
            and pass them to the log server
        end if
    case local shutdown
        terminate thread
    case config change:
        get new baseband control parameter from config db
        pass them to the MUBM
    }

    log thread shutdown
    set termination status

DATA: --

LOGIC: --

**** END MODULE HEADER ***
#include "BaseNt.h"
namespace US {
    // DATA:
    // INTERFACES:
    // void MUBM_Monitoring(
    //     // Arguments:
    //     // first parameter not used
    //     void* i_unused,
    //     // shutdown event
    //     const HANDLE i_shutdownEvent
    // );
    // Returns:           --
    // Exceptions:       --
}
#endif

4.2.6 US::CVCDU

**************************************************************************
****  DADF MODULE HEADER  ***
**************************************************************************

4-13
TYPE: Concrete Class

PURPOSE: Encapsulate VCDU processing

FUNCTION: Process each VCDU

REFERENCES: MSI

INTERFACES: See ‘INTERFACES:’ in the module declaration below

RESOURCES: Heap Memory (>2K)

REFERENCES:

PROCESSING: analyse the incoming stream data and extract the vcdu data field

DATA: See ‘DATA:’ in the class header below.

LOGIC:
< provided only where required >

**** END MODULE HEADER ***
***************************************************************************/
#include <sstream>
#include "GSDS_Volume_F.h"

namespace US
{
    class Cvcdu
    {

    public:
        // DATA:
        // INTERFACES:
        //
        Cvcdu();
        // Description: Constructor
        // Arguments: --
        // Returns: --
        // Exceptions: --

        bool Filler () { return (m_VcId == mc_vcIdFiller);} ;
        // Description: return if this is a filler vcdu
        // Arguments: --
        // Returns: true, if filler vcdu
        // Exceptions: --
unsigned __int32 GetBERCount() { return (m_BERCount);}
// Description: return if this is a filler vcdu
// Arguments: --
// Returns: true, if filler vcdu
// Exceptions: --

bool Bad () {return m_bad;};
// Description: vcdu is inconsistent and shall be ignored
// Arguments: --
// Returns: true, vcdu ok, else false
// Exceptions: --

bool ChainReset() {return m_reset;};
// description: transmission chain resetted or channel changed
// Arguments: --
// Returns: true, if reset detected
// Exceptions: no specific

unsigned int Lost () {return m_lost;};
// Description: Lost VCDUS between the last processed and this
// Arguments: --
// Returns: --
// Exceptions: --

unsigned In_avail () { return (m_pDataEnd - m_pData);}
// Description: No bytes available in the vcdu buffer
// Arguments: --
// Returns: true, if data zone contains the beginning of a source packet
// Exceptions: no specific

bool SyncToHeader();
// Description: Ignore data until beginning of next source packet and assign
// to current position
// Arguments: --
// Return: true, if data zone contains the beginning of a source packet
// Exceptions: no specific

int Read
// Description: Read data from the data zone beginning at the current position
{
  // Arguments:
  // read into buffer
  unsigned char *o_ReadBuffer,
  // buffer size in bytes ( default set to complete data zone)
  unsigned int i_ReadBufferSize,
  // flag indicating source packet continuation data
  bool i_continuationPacketData
};
// Returns: no of bytes read into the buffer
// Exceptions: --

void Reset ();
// Description: reset counter
// Arguments: --
// Returns: --
// Exceptions: --

// Get member functions
unsigned char GetVcId () {return m_VcId;};
unsigned GetVcduCounter () { return m_VcduChannelCounter;};
unsigned GetVcduFillerCounter () { return m_VcduFillerCounter;};

friend std::istream& operator>>(
    // Description: read vcdu data from stream
    
    // Arguments:
    std::istream &i_mubmVcduStream,
    Cvcdu &o_VcduData
    )
    // Returns: ref to std::istream
    // Exceptions: no specific
    
protected:
    // repeat the above for the protected data/functions

private:
    // DATA:
    // DATA:
    enum { mc_ImplVersionNo = 0x01}; // MSI 3.0
    enum { mc_vcIdFiller = 0x3F};
    enum { mc_SignalingField = 0};
    enum { mc_NoHeaderContained = 0x7FF};
    enum { mc_MinFirstHeaderPointer = 0};
    enum { mc_MaxFirstHeaderPointer = 883};
    // expected contents of VCDU header
    unsigned char m_ScId;
    // Virtual Channel Id for ...
    // processed data VCDUs
    unsigned char m_processedVC;
    // this vcdu
    unsigned char m_VcId;
    // vcdu counter
    unsigned m_VcduChannelCounter;
    unsigned m_VcduChannelCounterErrors;
    unsigned m_VcduFillerCounter;
    unsigned m_VcduFillerCounterErrors;
    // BER on filler vcdu
    unsigned short m_BERCount;
    // vcdu state
    // vcdu not usable
    bool m_bad;
    // transmission resetted
    bool m_reset;
    // lost count between last recevied one
    unsigned int m_lost;
    // M_PDU ( packed data unit)
    unsigned short m_first_header_pointer;
    // M_PDU packet zone
    unsigned char m_datazone[886];
// current data
unsigned char *m_pData;
unsigned char *m_pDataEnd;

// INTERFACE:
static std::string &GetScIdText
// Description: return the text representation of the space craft
// Arguments:
// space craft id
// i_scId
// Returns: pointer to the text
// Exceptions: no specific

};
} // end namespace US
#endif

4.2.7 US::CSourcePackets

>Type: Concrete Class
PURPOSE: Encapsulate Source packet processing

FUNCTION: Define processing and control data for a source packet

INTERFACES: See ‘INTERFACES:’ in the module declaration below

RESOURCES: --

REFERENCES: MSI

PROCESSING: read the data from the m_pdu and assemble source packets

DATA: See ‘DATA:’ in the class header below.

LOGIC:
< provided only where required >

**** END MODULE HEADER ***
**************************************************************************/
#include "CVCDU.h"
#include <fstream.h>
namespace US
{
    class CSourcePacket
    {
    public:
        // DATA:
        // Source packet sequence flags
        typedef enum
        {
            e_continuationSegment = 0,
            e_firstSegment = 1,
            e_lastSegment = 2,
            e_completeFile = 3,
            e_invalidFlag = 0x0f
        } ESequenceFlags;
        enum { e_SequenceMaskBegin = 0x01};
        enum { e_SequenceMaskEnd = 0x02 };
        typedef enum
        {
            e_streamHeader = 0x00,
            e_syncToHeader = 0x01
        } EHeaderSyncronisation;
        enum { e_apidMax = 0x3f};
        enum { e_apidFiller = 0x1f};
        enum { e_apidMaskHRIT = 0x20};
        enum { e_applicationDataLength = 8190};
// INTERFACES:

CSourcePacket ();
// Description: Constructor for the source packet
// Arguments: --
// Returns: --
// Exceptions: --

~CSourcePacket ();
// Arguments: --
// Returns: --
// Exceptions: --

bool Completed () {return m_dataCompleted;}
// Description: flag, indicating if source packet completed
// Arguments: --
// Returns: true, source packet data completed
// Exceptions: --

bool Ready () { return !m_dataCompleted && !m_SyncToHeader && !m_bad; }  
// Description: condition for a new source packet
// Arguments: --
// Returns: true, if source packet is new and have no data received
// Exceptions: false

bool Bad () { return m_bad;};
// Description: flag, indicating if the source packet is not usable
// Arguments: --
// Returns: value of m_bad
// Exceptions: --

bool SyncToHeader () {return m_SyncToHeader;}
// Description: flag, indicating if stream is lost and source packet
// has to synchronize to next header indicated by the header pointer
// in the VCDU.
// Arguments: --
// Returns: value of m_SyncToHeader
// Exceptions: --

// flags indicating type of source packet

bool FileContinuation () { return (m_sequenceFlags == e_continuationSegment) && !m_bad;}

bool FileBegin () { return ((m_sequenceFlags & e_SequenceMaskBegin) == e_firstSegment) && !m_bad;}

bool FileEnd () { return ((m_sequenceFlags & e_SequenceMaskEnd) == e_lastSegment) && !m_bad;}

bool CompleteFile () { return (m_sequenceFlags == e_completeFile) && !m_bad;}

bool Filler () { return ((m_apid & e_apidFiller) == e_apidFiller) && !m_bad;}

bool CRCCheck ();
// Description: build check sum and compare with attached one
// Arguments: --
// Returns: true, if checksum are the same and false else
// Exceptions: --

4-19
void InsertFiller
// Description: insert filler byte to sourcepacket
// Arguments:
// (unsigned int i_NoFillerBytes);
// Returns: --
// Exceptions: --

void Reset
// Description: reset the source packet contents
// Arguments:
// (EHeaderSyncronisation i_headerSync);
// Returns: --
// Exceptions: --

// INSERTION
CSourcePacket& operator<<
// Description: insert m_pdu data to source packet
// Arguments:
// Cvcdu &i_vcduData);
// Returns: this class
// Exceptions: no specific

int Read
// Description: read source packet data in databuffer
// Arguments:
// (unsigned char *o_dataBuffer,
// requested length int i_requestedDataLength,
// position of first corrupted byte in the data buffer unsigned short *o_corruptedPosition);
// Returns: no of bytes copied to the buffer
// Exceptions: no specific

int Read
// Description: read source packet and insert in the file stream
// Arguments:
// (ofstream o_fileStream,
// position of first corrupted byte in writen bytes unsigned short *o_corruptedPosition);
// Returns: no of bytes write to the file stream
// Exceptions: no specific

// SET member
void SetTimeStamp ( Util::CMSGTime i_timestamp)
{if (m_headerIndex == 0) { m_timeStamp = i_timestamp;});

void SetTimeStamp ( Util::CMSGTime i_timestamp)
{if (m_headerIndex == 0) { m_timeStamp = i_timestamp;});

void SetTimeStamp ( Util::CMSGTime i_timestamp)
{if (m_headerIndex == 0) { m_timeStamp = i_timestamp;});

// GET member
// reception time
Util::CMSGTime GetTimeStamp () { return m_timeStamp;};

// CRC result
bool GetCrcOk () { return m_crcOk;};

// no of corrupted data
UNSIGNED_SHORT GetNoCorruptedData () { return m_corruptedCount; });

// position of first corrupted bytes
INTEGER GetCorruptedPosition () { return m_corruptedPosition;}

// return CRC Check sum
UNSIGNED_SHORT GetCRCcheckSum () { return m_CrcCheckSum;}

// return data length
UNSIGNED_SHORT GetADFlength () { return m_adfLength; }

// return APDI
UNSIGNED_SHORT GetAPID () { return m_apid;};

// return sequence count
UNSIGNED_SHORT GetAPIDSequenceCount () { return m_sequenceCount;};

// Return raw data for tracing
void ReadRawHeader
// Description: Get header bytes
// Arguments:
// header byte field
// unsigned char o_SpHeader[6]
// Returns: header byte
// Exceptions: Util::CParamException, if i_index out of range

void ReadRawData
// Description: Get data field
// Arguments:
// buffer address
// unsigned char *o_spBuffer,
// I:sizeof buffer, 0:no of available bytes
// unsigned int *t_DataLength
// Returns: --
// Exceptions: Util::CParamException

protected:
// repeat the above for the protected data/functions

private:
// Data:

    // reception time stamp
    Util::CMSGTime m_timeStamp;
    // Source packet Header
    // temp. SP header storage
    unsigned char m_header[6];
    unsigned char m_headerIndex;
    bool m_headerCompleted;

    // ... Packet ID
    unsigned short m_apid;
    // ... Packet Sequence Control
    ESequenceFlags m_sequenceFlags;
    unsigned short m_sequenceCount;
    // ... Packet Length
    unsigned short m_packetLength;

    // Source Packet Data Field
    unsigned char m_adf[e_applicationDataLength+2];
    unsigned short m_adfLength;
    unsigned short m_readPosition;

    // CRC check values
    unsigned short m_CrcCheckSum;

    // completion flag
    bool m_dataCompleted;
    // no of corrupted byte within the sp data field
    unsigned short m_corruptedCount;
    unsigned short m_corruptedPosition;
    // result of crc check
    bool m_crcOk;

    // Assemble Controls
    bool m_bad;
    bool m_SyncToHeader;

};

} // end namespace US

#endif

4.2.8   US::CtransportFile

 ifndef _CTRANSPORTFILE_H_
 #define _CTRANSPORTFILE_H_
 
**************************************************************************
****  DADF MODULE HEADER   ***
**************************************************************************
**** DADF MODULE HEADER   ***

TYPE: Concrete Class
PURPOSE: Encapsulate Transport file processing

FUNCTION: Define the processing and data for transport file assembling.

INTERFACES: See INTERFACE description below

RESOURCES: --

REFERENCES: MSI

PROCESSING: Perform the assembling of Transport files from Source packets.

DATA: --

LOGIC: --

**** END MODULE HEADER ****
***************************************************************************/
#include "CSourcePacket.h"
#include <fstream.h>

namespace US
{

class CTransportFile
{
public:
    // DATA:
    // INTERFACES:
    // CTransportFile ();
    // Description: Constructor for a new Transport file
    // Returns: --
    // Exceptions: no specific
    ~CTransportFile ();
    // Description: Destructor
    // Arguments: --
    // Returns: --
    // Exceptions: no specific

    void Reset ();
    // Description: Reset file assembling
    // Arguments: --
    // Returns: --
    // Exceptions: no specific

}
void Release ();
// Description: Release the file assembling
// Arguments: --
// Returns: --
// Exceptions: no specific

void InsertFiller
// Description: Insert dummy bytes to data field
// Arguments:
// (No bytes to insert to the transport file
//   data field
//   unsigned int  i_noFillerBytes)
// Returns: --
// Exceptions: no specific

CTransportFile& CTransportFile::operator<<
// Description: insert application data field to transport file
// Arguments:
// (source packet data
//  CSourcePacket&  i_sourcePacket)

CTransportFile& CTransportFile::operator=
// Description: assign the descriptor information to another
// transport file
// Arguments:
// (transport file
//  CTransportFile& i_TransportFile)

bool Bad () { return m_bad;};
// Description: flag, indicating if the transport file is not usable
// Arguments: --
// Returns: value of m_bad
// Exceptions: --

bool Completed () { return (m_dataCompleted && !m_bad);};
// Description: flag, indicating if the assembling of the transport file
// has been completed.
// Arguments: --
// Returns: value of m_bad
// Exceptions: --

bool Empty () { return (m_headerIndex == 0);};
// Description: flag, indicating if file assembling has not yet begun
// Arguments: --
// Returns: true, if no header bytes are assembled
// Exceptions: --

// GET member
unsigned int GetFileCount () { return m_fileCounter;};
// Description: return the member file count
// Arguments: --
// Returns: contents of m_fileCount
// Exceptions: --

std::string GetFileSpec () { return m_xritFile;};
// Description: return the storage location of the transport file
// Arguments: --
// Returns: contents of m_xritFile
// Exceptions: --

Util::CMSGTime GetTimeStamp () { return m_timeStamp; }
// Description: return the assigned reception time
// Arguments: --
// Returns: msg time
// Exceptions: --

__int64 GetCorruptedCount () { return m_corruptedCount; }
// Description: return the count of the corrupted data
// Arguments: --
// Returns: byte count
// Exceptions: --

__int64 GetCorruptedBytePosition () { return m_corruptedPosition; }
// Description: return the byte position of the first corrupted byte
// Arguments: --
// Returns: byte position
// Exceptions: --

unsigned int GetAPID () { return m_apid; }
// Description: return the apid belonging to that file
// Arguments: --
// Returns: APID (see MSI)
// Exceptions: --

protected:
private:
// DATA:
// reception time stamp
Util::CMSGTime m_timeStamp;

// type checking
unsigned int m_apid;
// state
bool m_bad;

// temp. header storage until header assembling completed
unsigned char m_header[32];
unsigned char m_headerIndex;
bool m_headerCompleted;

// file counter
unsigned int m_fileCounter;
// aligned file length in bytes
__int64 m_fileLength;

// Transport file data field
bool m_dataCompleted;
unsigned int m_dataLength;

// get xrit header length from primary header
unsigned m_totalXRITHeaderLength;
__int64 m_XRITDataFieldLength;

// XRT file buffer
std::string m_xritFile;
ofstream m_xritFileStream;

// error handling for xrit buffer
__int64 m_corruptedCount;
__int64 m_corruptedPosition;

};
} // end namespace US

#endif

4.3 AdminOnlineStorage
4.3.1 US::AdminOnlineStorage

/*****************************************************************************/
* Eumetsat Meteosat Second Generation
* Data Acquisition and Dissemination Facility: DADF
* This software is developed by VCS Nachrichtentechnik GmbH under contract
* VCS : V97.103.1008
* Eumetsat : EUM/CO/97/499/YB
* Copyright VCS 1998
* Applicable Third-party Software Licence Information:
* *--------------------------------------------------------------
* Configuration Control Details from Visual Source Safe
* $Archive: $
* $History: $
* *--------------------------------------------------------------
* *******************************************
*/

#ifndef _US_ADMINONLINEBUFFER_H_
#define _US_ADMINONLINEBUFFER_H_
/***************************************************************************/
 ***  DADF MODULE HEADER   ***
 TYPE: THREAD
PURPOSE: Take LRIT/HRIT files received by the US::FileAssembler and control the further processing

FUNCTION: Get description of transport files from the queue and prepare the further processing.

INTERFACES: Using Util::CThread for the thread creation requires the Interface defined below. See ’INTERFACES:’ in the module declaration below.

RESOURCES: --

REFERENCES: --

PROCESSING: reads the description parameter for a received file from a queue create an xritprocessing object and pass the file. The xrit processing object analyse the file and determines the further processing for that file. If the processing is completed, the xrit processing object is passed to US::PassXritToDise and US::AutoTransferServer for transfer to the network and to the US::QuicklookServer, if the xrit contains image data.

DATA:

LOGIC: --

**** END MODULE HEADER ***
***************************************************************************/
#include "BaseNT.h"
#include <string>
namespace US
{
    // DATA:

    // INTERFACES:

    // Assembling of VCDUs to LRIT/HRIT Files
    // --------------------------------------

    void US_AdminOnlineBuffer(
    // Arguments:                  // Parameter not needed for function, but for CThread
    int* i_notUsed,
    // Shutdown event
    const HANDLE i_ShutdownEvent
    );

    // Returns: --

    // Exceptions: --
}
#ifndef _CXRITPROCESSING_H_
#define _CXRITPROCESSING_H_
/**************************************************************************
 ****  DADF MODULE HEADER     ***
**************************************************************************

TYPE: Concrete Class

PURPOSE: administrate the xrit file processing

FUNCTION: containing controlling parameter for the xrit file processing

INTERFACES: See ‘INTERFACES:’ in the module declaration below

RESOURCES: --

REFERENCES: --

PROCESSING: get a file and check, if its contents is usable as a xrit file process the file ( decryption and decompression) if neccessary.
save processing result

DATA: See ‘DATA:’ in the class header below.

LOGIC:
<!-- provided only where required -->

**** END MODULE HEADER   ***
**************************************************************************/
#include "CXritFile.h"
#include "CSKUControl.h"
#include "CThread.h"
#include "TimeUtil.h"
#include "CxRITFileDecompressed.h"
#include "CxRITFileDecrypted.h"

namespace US
{
class CXritProcessing
{
public:
// DATA:
enum EXritFileState
{
  e_Encrypted = 0,
  e_Compressed = 1,
  e_Clear = 2
};
#pragma pack(push, 1)
struct SEncryptionKey
{
  NBO::UNSIGNED_SHORT m_skuNumber;
  unsigned char m_keyNumber;
  unsigned char m_publicKey[24];
  NBO::UNSIGNED_SHORT m_pbkCRC;
};
#pragma pack ( pop)
// INTERFACES:
// CXritProcessing ();
// Description: default constructor
// Arguments: --
// Returns: --
// Exceptions: --
CXritProcessing
// Description: constructor used for an xrit file
// Arguments:
  (std::string &i_XritFileSpecification)
// Returns: --
// Exceptions: --
CXritProcessing

{  
  CXritProcessing *ip_refProcessing
};

~CXritProcessing ();
// Description: destructor
// Arguments: --
// Returns: --
// Exceptions: --

static void CXritProcessing::Start
// Description: Start the processing as seperatred thread
// using CThread
// Arguments:
// the Xrit processing
CXritProcessing *i_pXritProcess,
// shutdown event
const HANDLE i_shutdownEvent
);
// Returns: --
// Exceptions: no specific

void CXritProcessing::RunXritProcess
// Description: process a received XRT file
// Arguments:
// shutdown event
HANDLE i_shutdownEvent
);
// Returns: nothing
// Exceptions: no specific

bool SaveToFile
// Description: save processed data to file
// Arguments:
// xrit contents
DISE::CxRITFile *i_xritFile
);
// Returns: true, if successfully completed
// else false
// Exceptions: no specific

bool TakeReceivedFile
// Description: copy the received file to the online buffer location
// Arguments:
// source file specification
std::string &i_xritNameReceived
);

bool RemoveFiles ()
// Description: remove all files of this processing objectz
// Arguments: --
// Returns: --
// Exceptions: --
// Status informations
bool Good () { return m_ContinueProcessing;};

bool CompressedFileAvail() { return m_xritNameCompressed.size() > 0;};
bool ClearFileAvail() { return m_xritNameClear.size() > 0;};
bool EncryptedFileAvail(){ return m_xritNameEncrypted.size() > 0;};

// file types
bool EKMFileType () { return (m_FileTypeCode == DISE::e_EncryptionKeyMessage);};
bool ImageFileType () { return (m_FileTypeCode == DISE::e_ImageDataFile);};

// Processing
bool LoadEKMFile ();
// Description: interpret the contents of the data field as
// encryption key messages and load them to sku
// Arguments: --
// returns: --
// Exceptions: --

bool DecryptFile
// Description: decrypt an encrypted file
// Arguments:
// encrypted file
DISE::CxRITFile *i_encryptedFile,
// decrypted file
std::auto_ptr <DISE::CxRITFileDecrypted>& o_decryptedFile
);
// Return: true after successfully completion, else false
// Exceptions: no specific

bool DecompressFile
// Description: decompress a compressed file
// Arguments:
// compressed xrit file
DISE::CxRITFile *i_compressedFile,
// decompressed xrit file
std::auto_ptr <DISE::CxRITFileDecompressed>& o_decompressedFile
);
// Return: true after successfully completion, else false
// Exceptions: no specific

void SetTimeStamp ( Util::CMSGTime& i_timeStamp )
{ m_timeStamp = i_timeStamp;};

void SetReceptionQuality
// Description: set reception quality
// Arguments:
// __int64 &i_noCorruptedBytes,
// __int64 &i_lstCorruptedPosition
);
// Returns: --
// Exceptions: --

void SetProcessId ( Util::CThread<US::CXritProcessing>* i_tptrProcessing)
HANDLE GetProcessingHandle () {
    // Description:  get thread handle
    // Arguments:  --
    // Returns:  handle of processing thread
    // Exceptions:  --

    return;
}

std::string GetFileSpecAsReceived () {
    switch (m_StateAsReceived) {
        case e_Encrypted:
            return m_xritNameEncrypted;
            break;
        case e_Compressed:
            return m_xritNameCompressed;
            break;
        case e_Clear:
            return m_xritNameClear;
            break;
        default:
            return (std::string("unknown file"));
            break;
    }
}

EXritFileState GetFileTypeAsReceived () { return m_StateAsReceived;};

int GetSpectralChannelID () { return m_SpectralChannelId;};
__int8 GetPercentReceived () { return m_ReceptionResult; };  
__int8 GetPercentDecrypted () { return m_DecryptionResult; };  
__int8 GetPercentDecompressed () { return m_DecompressionResult; };  

DISE::CxRITAnnotation& GetReceivedAnnotation () { return m_xritAnnotationAsReceived;};
Util::CMSGTime& GetReceptionTime () { return m_timeStamp;};
const std::string GetClearSpec () { std::string o_value = m_ProductPath + m_xritNameClear; return o_value;}
const std::string GetCompressedSpec () { std::string o_value = m_ProductPath + m_xritNameCompressed; return o_value;}
const std::string GetEncryptedSpec () { std::string o_value = m_ProductPath + m_xritNameEncrypted; return o_value;}

// init globals
static void Init () {
    // Description:  init all globals after setup
    // Arguments:  --
    // Returns:  --
    // Exceptions:  --

    protected:

        // repeat the above for the protected data/functions

private:
// DATA:
// performance
SYSTEMTIME m_beginTime;
SYSTEMTIME m_endTime;
// reception / creation time
Util::CMSGTime m_timeStamp;
// processing control
bool m_ContinueProcessing;
bool m_notUsable;
// naming
std::string m_ProductPath;

// quality information of the received data field
__int64 m_noCorruptedBytes;
__int64 m_1stCorruptedByte;
unsigned short m_ReceptionResult;

// which type of file is received
EXritFileState m_StateAsReceived;
DISE::EFileTypeCode m_FileTypeCode;
std::auto_ptr <DISE::CxRITFileHeaderRecords>m_xritAsReceived;
DISE::CxRITAnnotation m_xritAnnotationAsReceived;
int m_SpectralChannelId;

// clear xrit file
std::string m_xritNameClear;

// compressed xrit file
std::string m_xritNameCompressed;
bool m_DecompressionRequested;
std::vector< __int16> m_LineQualityInfo;
unsigned short m_DecompressionResult;

// encrypted xrit file
std::string m_xritNameEncrypted;
bool m_DecryptionRequested;
unsigned short m_DecryptionResult;
// pseudo noise key for decryption
unsigned __int64 m_PseudoNoiseKey[3];

// transfer flags
Util::CThread<US::CXritProcessing>* m_tptrProcessing;

// Station key unit device ( setup by init function
static CSKUControl *m_skuDevice;

};
} // end namespace US

#endif
4.3.3  US::QuicklookServer

/**************************************************************************
* Eumetsat Meteosat Second Generation
* Data Acquisition and Dissemination Facility: DADF
* This software is developed by VCS Nachrichtentechnik GmbH under contract
* VCS : V97.103.1008
* Eumetsat : EUM/CO/97/499/YB
* Copyright VCS 1998
* Applicable Third-party Software Licence Information:
* Configuration Control Details from Visual Source Safe
* $Archive: $
* $History: $
*------------------------------------------------------------------------

#ifndef _QUICKLOOKSERVER_H_
define _QUICKLOOKSERVER_H_
/**************************************************************************
****  DADF MODULE HEADER   ***

TYPE: THREAD

PURPOSE: Serves the sub sampling and monitoring of image data

FUNCTION: Receives XRIT files of type image data from the
US::AdminOnlineProcess.
   The data are subsampled to a size fitting to the quicklook display. The
sub
   sampled packets are passed through the monitoring mmi via a generic
server.
   Additionally they are saved to a file for the historical quicklook.

INTERFACES: Using Util::CThread for the thread creation requires
the Interface defined below.
   See ‘INTERFACES:’ in the module declaration below

RESOURCES: --

REFERENCES: --

PROCESSING:
DATA:

LOGIC: --

**** END MODULE HEADER  ***
**************************************************************************/
#include "BaseNT.h"
#include <string>
namespace US
{
  // DATA:
  
  // INTERFACES:
  
  // Serves the subsampling and monitoring of image data
  // ---------------------------------------------------
  void QuicklookServer
  (  
    // Arguments:
      // first parameter not used
    void *i_unused,
    *   // shutdown event
    const HANDLE i_shutdownEvent  
  );
  //
  // Returns: --
  //
  // Exceptions: --
  
  #endif

4.3.4  US::Auto Transfer
**************************************************************************/
/* Eumetsat Meteosat Second Generation
* Eumetsat Meteosat Second Generation
* Data Acquisition and Dissemination Facility: DADF
* This software is developed by VCS Nachrichtentechnik GmbH under contract
* VCS : V97.103.1008
* Eumetsat : EUM/CO/97/499/YB
* Copyright VCS 1998
* Applicable Third-party Software Licence Information:
* Configuration Control Details from Visual Source Safe
* $Archive: $
* $History: $
* 
* ***************************************************************************/
#ifndef _QUICKLOOKSERVER_H_
#define _QUICKLOOKSERVER_H_
/**************************************************************************
****  DADF MODULE HEADER   ***
TYPE: THREAD

PURPOSE: Serves the transfer of processed xrit files depending on the filter

FUNCTION: Receives information about received products from the US::AdminOnlineProcess.
Search for a matching filter entry and if a transfer is requested, then pass to the selected host.

INTERFACES: Using Util::CThread for the thread creation requires the Interface defined below.
See 'INTERFACES:' in the module declaration below

RESOURCES: --

REFERENCES: --

PROCESSING:

DATA:

LOGIC: --

****  END MODULE HEADER   ***
**************************************************************************/
#include "BaseNT.h"
#include <string>
namespace US
{
  // DATA:

  // INTERFACES:
  //
  // Serves the transfer of processed file to the network
  // ----------------------------------------------------
  void AutoTransfer_Server
  {
    // Arguments:
    // first parameter not used
    void *i_unused,
    // shutdown event
    const HANDLE i_shutdownEvent
  );

    // Returns: --
    //
    // Exceptions: --
  }
#endif
4.4 PassXRITDataToDise

4.4.1 US::PassXRITDataToDise

/****************************************************************************
* Eumetsat Meteosat Second Generation
* Data Acquisition and Dissemination Facility: DADF
* This software is developed by VCS Nachrichtentechnik GmbH under contract
* VCS : V97.103.1008
* Eumetsat : EUM/CO/97/499/YB
* Copyright VCS 1998
* Applicable Third-party Software Licence Information:
* ---------------------------------------------------------------------------
* Configuration Control Details from Visual Source Safe
* $Archive: $ *
* $History: $ *
* *---------------------------------------------------------------------------
* ***************************************************************************/

#ifndef _US_XRITTODISE_H_
define _US_XRITTODISE_H_
/****************************************************************************
**** DADF MODULE HEADER ****

TYPE: THREAD

PURPOSE: Send monitoring and file data to DISE

FUNCTION: Get the processing information from the US::US_AdminOnlineBuffer
create the monitoring data and pass them to DISE
filter the file data and pass them to DISE

INTERFACES: Using Util::CThread for the thread creation requires
the Interface defined below.
See 'INTERFACES:' in the module declaration below

RESOURCES: --

REFERENCES: --

PROCESSING:
DATA:

LOGIC: --

**** END MODULE HEADER ***
**************************************************************************
#include "BaseNT.h"
#include <string>
#include <vector>
namespace US {
// DATA:
// INTERFACES:
// // Assembling of VCDUs to LRIT/HRIT Files
// // Assembling of VCDUs to LRIT/HRIT Files
void US_XritToDise
{
    // Arguments:
    void* i_unused,
    const HANDLE i_shutdownEvent
;}
// Returns: --
// Exceptions: --

static DWORD ReadIntervalParameter
// Description: read the interval parameter specified by the
// configuration db path of transfer to dise part
// from the registry and check against the ranges
// Arguments:
{
    class CConfigParams &i_configParameter,
    const std::string &i_confValDiseInterval
};
// Returns: the interval
// Exceptions: no specific

static DWORD ReadMonBufferTime
// Description: read the monitoring buffer time from the registry and
// check against the ranges
// Arguments:
{
    class CConfigParams &i_configParameter
};
// Returns: buffer time of monitoring data in minutes
// Exceptions: no specific

static void ExchangeHandle
// Description: exchange handle in a handle list
// Arguments:
handle list
    std::vector<HANDLE> &i_handleList,
    HANDLE i_findHandleValue,
    HANDLE i_replaceHandleValue

    // Returns:
    // Exceptions: Util::CNamedException, if findHandleValue not found in list
#

## 4.4.2 US::PassMonitoringData

```cpp
// handle list
std::vector<HANDLE> &i_handleList,
    HANDLE i_findHandleValue,
    HANDLE i_replaceHandleValue

    // Returns:
    // Exceptions: Util::CNamedException, if findHandleValue not found in list

#endif
```

### 4.4.2 US::PassMonitoringData

/*******************************************************************************
* Eumetsat Meteosat Second Generation
* Data Acquisition and Dissemination Facility: DADF
* This software is developed by VCS Nachrichtentechnik GmbH under contract
* VCS      : V97.103.1008
* Eumetsat : EUM/CO/97/499/YB
* Copyright VCS 1998
* Applicable Third-party Software Licence Information:
*------------------------------------------------------------------------------
* Configuration Control Details from Visual Source Safe
* $Archive:
* $History:
*------------------------------------------------------------------------------
*--------------------------------------------------------------------------------
*/

#ifndef _CMONITORINGSERVER_H_
#define _CMONITORINGSERVER_H_

/**************************************************************************
**  DADF MODULE HEADER   ***
TYPE: Concrete Class
PURPOSE: Encapsulate transfer of monitoring data to DISE
FUNCTION: pass monitoring information of the received files from the US to the DISE
INTERFACES: See ‘INTERFACES:’ in the module declaration below

**** DADF MODULE HEADER ***

TYPE: Concrete Class

PURPOSE: Encapsulate transfer of monitoring data to DISE

FUNCTION: pass monitoring information of the received files from the US to the DISE

INTERFACES: See ‘INTERFACES:’ in the module declaration below
```
RESOURCES: heap memory for queue

REFERENCES: US::DISE-ICD

PROCESSING: Create an instance of a COMS::CBaseServerComms to provide the server side of the US::DISE monitoring transfer. The monitoring data are buffered in a COMS::CTimeBasePurgedAQLQueue.

DATA: See ‘DATA:’ in the class header below.

LOGIC:

**** END MODULE HEADER ***
**************************************************************************/

// COMMS includes
#include "DADFIntRegister.h"
#include "CInternalCommsLinkProtocol.h"
#include "CBaseServerComms.h"
#include "CMSGEndPoint.h"
#include "CDADFPacketFunctions.h"
#include "Ports.h"
#include "CTimeBasedPurgingALQ.h"

// Packet lib
#include "US_EventPackets.h"
#include "LogDefinitions.h"
// DISE
#include "CxRITStatus.h"

namespace US
{
    class CMonitoringObject : public COMS::CTimeBasedPurgingObject
    {
        public:
        CMonitoringObject
        // Description: create a xrit status object
        // Arguments:
        {
        annotation header of xrit file
        DISE::CxRITAnnotation &i_annotationHeader,
        // percent of xrit data received
        unsigned __int8 i_ReceptionPercent,
        // percent of xrit data decrypted
        unsigned __int8 i_DecryptionPercent,
        // percent of xrit data decompressed
        unsigned __int8 i_DecompressionPercent,
        // time reception of the file starts
        Util::CMSGTime i_ReceptionTime
        );
        // Returns: --
        // Exceptions: no specific
// function called by queue
void PacketDelivered();
void PacketPurged();
std::auto_ptr<COMS::CDADFMessagePacket> GetPacket()
{
    DISE::CxRITStatus temp (m_annotationHeader,m_monInfo);
    std::auto_ptr<COMS::CMSGBaseNetworkClass> aptemp (temp.Clone());
    std::auto_ptr<COMS::CDADFMessagePacket> apMsgPacket =
    COMS::CreateInternalMessagePacket(temp.Clone());
    return apMsgPacket;
};
const Util::CMSGTime &QueueTime() { return m_queuedTime;};

private:
    std::auto_ptr<DISE::CxRITStatus> m_aptrMonitoringData;
    DISE::CxRITAnnotation m_annotationHeader;
    DISE::CMonitoringInfoUS m_monInfo;
    
    Util::CMSGTime m_queuedTime;
);

class CMonitoringServer
{
public:
    // DATA:
    // INTERFACES:
    // CMonitoringServer
    // Description: constructor of the server
    // ip Address of the server
    std::string &i_ipAddress,
    // unsigned short
    unsigned short i_portNo,
    // time the monitoring data shall be buffered
    Util::CMSGTimeSpan &i_monBufferTime
    );
    // Returns: --
    // Exceptions: no specific

    ~CMonitoringServer();
    // Description: destructor
    // Arguments. --
    // Returns: --
    // Exceptions: --

    void Stop();
    // Description: stop the running server and queue
    // Arguments: --
    // Returns: --
    // Exceptions: --
void ResetBufferTime
// Description: set new time the monitoring data shall be buffered
// Arguments:
{
    Util::CMSGTimeSpan &i_monBufferTime
}
    m_aptrMonBuffer->ResetPurge ( i_monBufferTime);return;}
// Returns: --
// Exceptions: --

void AddMonitoringDataToBuffer
// Description: add monitoring data to buffer
// Arguments:
{
    DISE::CxRITAnnotation &i_annotationHeader,
    unsigned __int8 i_ReceptionPercent,
    unsigned __int8 i_DecryptionPercent,
    unsigned __int8 i_DecompressionPercent,
    Util::CMSGTime &i_ReceptionTime
};

HANDLE GetConnectedEvent () { return mp_ConnectedEvent->get();};
// Description: return handle indicate connection established
// Arguments: --
// Returns: handle of event
// Exceptions: no specific

HANDLE GetDisconnectedEvent () { return mp_DisconnectedEvent->get();};
// Description: return handle indicate disconnection from dise
// Arguments: --
// Returns: handle of event
// Exceptions: no specific

protected:

// repeat the above for the protected data/functions

private:

// DATA:
// Build an end point class
Util::DADFLinkId *mp_EndPoint;

// The Register of functions
std::auto_ptr<COMS::CIntRegister> m_apRegister;
// protocol class
COMS::CInternalCommsLinkProtocol *mp_linkProtocol;

// Create the server instance.
COMS::CBaseServerComms *mp_server;

// events indicating the link state
Util::CMSEvent *mp_ConnectedEvent;
Util::CMSGEvent *mp_DisconnectedEvent;

    // time base purging queue
    std::auto_ptr<COMS::CTimeBasedPurgingALQ> m_aptrMonBuffer;

};
} // end namespace US
#endif

4.4.3 US::PassFileData

/****************************************************************************
** Eumetsat Meteosat Second Generation
** Data Acquisition and Dissemination Facility: DADF
** This software is developed by VCS Nachrichtentechnik GmbH under contract
** VCS      : V97.103.1008
** Eumetsat : EUM/CO/97/499/YB
** Copyright VCS 1998
** Applicable Third-party Software Licence Information:
**------------------------------------------------------------------------------
** Configuration Control Details from Visual Source Safe
** $Archive:
**$History:
**------------------------------------------------------------------------------
**
#endif _CFILEDISESERVER_H_
#define _CFILEDISESERVER_H_
/****************************************************************************
**** DADF MODULE HEADER   ***

TYPE:    Concrete Class

PURPOSE:  Encapsulate transfer of data data to DISE

FUNCTION:  pass the content of received/processed files from the
US to the DISE

---
INTERFACES: See ‘INTERFACES:’ in the module declaration below

REFERENCES: US::DISE-ICD

PROCESSING: Create an instance of a COMS::CBaseServerComms to provide the server side of the US::DISE file transfer.

DATA: See ‘DATA:’ in the class header below.

LOGIC:

**** END MODULE HEADER ****
**************************************************************************/
// COMMS includes
#include "DADFIntRegister.h"
#include "CInternalCommsLinkProtocol.h"
#include "CBaseServerComms.h"
#include "CMSGEndPoint.h"
#include "CDADFPacketFunctions.h"
#include "Ports.h"

// Packet lib
#include "US_EventPackets.h"
#include "LogDefinitions.h"
// DISE
#include "CxRITStatus.h"

namespace US
{

class CFileDiseServer
{
public:
// DATA:
// INTERFACE:
// CFileDiseServer
// Description: constructor of the server
{
// ip Address of the server
std::string &i_ipAddress,
// unsigned short
unsigned short i_portNo
};
// Returns: --
// Exceptions: no specific

-CFileDiseServer();
// Description: destructor
// Arguments. --
// Returns: --
// Exceptions: --

void Stop ();
// Description: stop the running server and queue
// Arguments: --
// Returns: --
// Exceptions: --

void PutFile
// Description: pass file data to comm server
// Arguments:
{
  // xritfile spec
  std::string i_xritFileSpec
};
// Returns: --
// Exceptions: --

HANDLE GetConnectedEvent () { return mp_ConnectedEvent->get();};
// Description: return handle indicate connection established
// Arguments: --
// Returns: handle of event
// Exceptions: no specific

HANDLE GetDisconnectedEvent () { return mp_DisconnectedEvent->get();};
// Description: return handle indicate disconnection from dise
// Arguments: --
// Returns: handle of event
// Exceptions: no specific

protected:

// repeat the above for the protected data/functions

private:

// DATA:
// // Build an end point class
Util::DADFLinkId *mp_EndPoint;
// // The Register of functions
std::auto_ptr<COMS::CIntRegister> m_apRegister;
// // protocol class
COMS::CInternalCommsLinkProtocol *mp_linkProtocol;
// // Create the server instance.
COMS::CBaseServerComms *mp_server;
// // events indicating the link state
Util::CMSGEvent *mp_ConnectedEvent;
Util::CMSGEvent *mp_DisconnectedEvent;
4.5 Manual Decryption

4.5.1 US::ManualDecryption

#pragma once

/**************************************************************************
***  DADF MODULE HEADER   ***
TYPE: Process
PURPOSE: MMI started decryption
FUNCTION: open a LRIT/HRIT file, decrypt it and store the result in a file.
INTERFACES: See ‘INTERFACES:’ in the module declaration below
RESOURCES:
REFERENCES: --
PROCESSING: opens the encrypted LRIT/HRIT file and map to memory
read keyHeaderRecord and ask the SKUControl for a PN-key assigned to the key and seed.
creates a file taking the decrypted LRIT/HRIT data and map it to memory.
copy the header field except the key header record from the encrypted to the decrypted file.
set encryptedData to start of data field (encrypted file)
set decryptedData to start of data field (decrypted file)
find a suitable PN-pattern length
for ( WorkCount = 0; WorkCount < DataFieldSize; Workcount + PN-patternlength)
{
call PNGenerator ( PN_Pattern PN-patternlength);
decryptedData = bit-wise addition of encryptedData and PN-pattern

*/
WorkCount = WorkCount + Patternlength

Close encrypted file and store decrypted file to disk

DATA:

LOGIC:

**** END MODULE HEADER ***
**************************************************************************

// DATA:

// INTERFACES:
//
// void main ( int argc,
//             unsigned char argv[])
//
// Arguments:
//            argv[0] encryptedXritFile,
//            argv[1] decryptedXritFile
//
// Returns: --
// Exceptions: --

4.6 Manual Decompression

4.6.1 US::ManualDecompression

#pragma once
/**************************************************************************
**** DADF MODULE HEADER ****
**************************************************************************

TYPE:    thread

PURPOSE:   MMI start decompression of LRIT/HRIT files

FUNCTION:  Decompress the data field of a compressed LRIT/HRIT file
            and store the result to disk

INTERFACES: See ’INTERFACES:’ in the module declaration below
RESOURCES: DADF-COMP

REFERENCES: MSI

PROCESSING: opens the compressed LRIT/HRIT file and map to memory (compressedFile).
calculate the data size of the decompressed file
creates a file taking the decompressed LRIT/HRIT data and map it to memory (decompressedFile).
copy the header field from the compressed to the decompressed file.
reset the Image_Representation in the ImageSegmentIdentification and the Compression_Flag in the ImageStructure-record of the decompressedFile
Update the Data_Field_Length of the PrimaryHeader to the decompressed size
setup parameter for decompression function
start decompression

Store decompressed file to disk
set termination status;

DATA: See ‘DATA:’ in the class header below.

LOGIC: --

**** END MODULE HEADER ***
**************************************************************************/

// DATA:

// INTERFACES:
//
// void main ( int argc, char argv[] );
// Arguments:
// argv[0] compressedXritFile, argv[1] decompressedXritFile
// Returns: --
// Exceptions: --
4.7 Common Components

4.7.1 US::Cmubm_IF

/*************************************************************
 **** DADF MODULE HEADER ****

TYPE: Base Class

PURPOSE: Interfacing the User station PC to the external MUBM

FUNCTION: Implement the SCSI commands defined by the MUBM interface

INTERFACES: See ’INTERFACES:’ in the module declaration below

RESOURCES:

REFERENCES: SCSI Standard-II
ICD MUBM::Workstation [USDDS-B Annex-A]

PROCESSING: To communicate with the MUBM device, the following steps are necessary:
- connect to the SCSI port adapter by the Open-Function
- pass commands for the MUBM device via the SCSI interface (see implemented commands)
- disconnect from the SCSI port adapter by the close -command

DATA: --

LOGIC: --

**** END MODULE HEADER ****
*************************************************************/
#endif
#define CMUBM_IF_H
#include <string.h>
#include "BaseNT.h"
#include "ErrorHandling.h"
#include "GSDS_Volume_F.h"
#include "NTDDSCSI.H"
#include "DEVIOCTL.h"
#include "ld_ld.h"
namespace US
{

class CMUBMDev
{
public:
    // DATA:
    typedef enum
    {
        e_lun0 = 0,
        e_lun1 = 1,
        e_lun2 = 2,
        e_lun3 = 3,
        e_lun4 = 4,
        e_lun5 = 5,
        e_lun6 = 6,
        e_lun7 = 7
    } ELogicalUnit;

    typedef enum
    {
        e_RBufferIdVCDU = 0x80,
        e_WBufferIdSKU = 0x83,
        e_RBufferIdSKU = 0x84,
        e_WBufferIdCtrl = 0x85,
        e_RBufferIdMon = 0x86,
        e_RBufferIdLog = 0x87
    } EBufferId;

    typedef enum
    {
        e_NoDisconnect = 0,
        e_DisconnectFor3s = 1,
        e_DisconnectFor10s = 2,
        e_DisconnectFor30s = 3
    } EBufferOffset;

    // INTERFACES:
    //
    // SCSI Adapter
    static DWORD Open
    ( // Arguments:
        std::string& i_portName,
        unsigned char i_busId,
        unsigned char i_targetId,
        unsigned char i_SupportedLUNs
    );

    // Returns: system status describing the result of the device open operation
static bool Close()
// Description: Cancel queued I/Os
// Arguments: --
// Returns: system status code
// Exceptions: --

static void Cancel
// Description: returns the sense data available for this logical unit
// Arguments: --
// Returns: sense data code
// Exception: no

static std::string SenseMsg
// Description: return the sense message available for this logical unit
// Arguments: --
// Returns: sense data message
// Exception: no

static bool LunReady
// Description: test logical unit ready incl. reset handling
// Arguments: --
// Returns: sense data message
// Exception: no
// IMPLEMENTATION of MUBM Command set
// ==================================================
static EScsiStatus TestUnitReady
// Description: Test, if a logical unit of the MUBM is ready
// Arguments:
// select the logical unit to be tested
// i_logicalUnit
// Returns: return the SCSI status value:
// Exceptions: --

static EScsiStatus RequestSense
// Arguments:
// select the logical unit
// i_logicalUnit,
// Sense
// o_sense
// Returns:
// Exceptions: --
// INQUIRY COMMAND
static EScsiStatus Inquiry
// Arguments:
// select the logical unit
// i_logicalUnit,
// inquiry information
// o_inquiryData
// Returns: SCSI status returned by command completion
// Exceptions: --

static EScsiStatus SendDiagnostic
// Arguments:
// i_logicalUnit select the logical unit
// o_SelfTestFailed Result of the selftest
// Returns:
// Exceptions: --

static EScsiStatus GetMessage
// ELogicalUnit i_logicalUnit,
void *o_MessageBuffer,
DWORD  i_MessageBufferSize
);
// Arguments:
// i_logicalUnit select the logical unit
// o_MessageBuffer returning the message received by the MUBM
// i_MessageBufferSize determining the size of the message buffer
// o_MessageLength returning the length of the received message
//
// Returns:
// system status describing the result of the request sense command
//
// Exceptions: --

static EScsiStatus SendMessage
{
    ELogicalUnit i_logicalUnit,
    void     *i_MessageBuffer,
    DWORD    i_MessageLength
);
// Arguments:
// i_logicalUnit select the logical unit
// i_MessageBuffer containing the message send to the MUBM
// i_MessageLength length of the send message
//
// Returns: system status describing the result of the request sense command
//
// Exceptions: --

static EScsiStatus WriteBuffer
// Description: synchronous write buffer command
{
    ELogicalUnit i_logicalUnit,
    EBufferId    i_bufferId,
    EBufferOffset i_bufferOffset,
    unsigned     i_bufferLength,
    void        *i_buffer,
    unsigned     i_bufferTransferred
    void        *o_bufferTransferred
);
// Returns: MUBM target scsi status
//
// Exceptions: no specific

static EScsiStatus ReadBuffer
// Description: synchronous read buffer command
{
    ELogicalUnit i_logicalUnit,
// specifies the buffer to read
   EBufferId   i_bufferId,
// specifies the disconnect time
   EBufferOffset i_bufferOffset,
// specifies the length of the buffer
   unsigned   i_bufferLength,
// buffer to read in data
   void *o_buffer,
// count of transferred bytes
   unsigned   *o_bufferTransferred
);

// Returns: MUBM target scsi status
// Exceptions: no specific

static EScsiStatus ReadBuffer
// Description: asynchronous read buffer command
{
// Arguments:
// specify the logical unit
   ELogicalUnit i_logicalUnit,
// specifies the buffer to read
   EBufferId   i_bufferId,
// specifies the disconnect time
   EBufferOffset i_bufferOffset,
// specifies the length of the buffer
   unsigned   i_bufferLength,
// buffer to read in data
   void *o_buffer,
// command completion event
   HANDLE *o_CompletionEvent
);

// Returns: MUBM target scsi status
// Exceptions: Util::CNTException
// Util::CNamedException

static EScsiStatus GetResult
// Description: get result of asynchronous operation after
// completion event set
{
// Arguments:
   // logical unit number
   ELogicalUnit i_logicalUnit,
   // transfer count
   DWORD *o_transferCount
);

// Returns: scsi status of completed operation
// Exceptions: Util::CNTException

protected:

private:
    //INTERFACE:
CMUBMDev();
// Arguments: --
// Returns: --
// Exceptions: --

// Returns: data returned are valid
// Exceptions: --

// Data:
// Length of MUBM fixed data fields
enum {e_SenseInfoLen = 14};
// specification of SCSI host adapter
static std::string m_portName;
// handle of open SCSI host adapter
static HANDLE m_portHandle;
// SCSI bus ID
static UCHAR m_busId;
// SCSI Target ID
static UCHAR m_targetId;
// SCSI logical unit list
static CLUNDescriptor m_logicalUnit[8];
};
} // end namespace US
#endif

4.7.2 US::CconfigParams

/******************************
* Eumetsat Meteosat Second Generation
* Data Acquisition and Dissemination Facility: DADF
* This software is developed by VCS Nachrichtentechnik GmbH under contract
* VCS : V97.103.1008
* Eumetsat : EUM/CO/97/499/YB
* Copyright VCS 1998
* Applicable Third-party Software Licence Information:
* Configuration Control Details from Visual Source Safe
* $Archive: /US Element/US/US_Functions/inc/CConfigParams.h $
* $History: CConfigParams.h $
* 
*
#ifndef _CCONFIGPARAMS_H_
define _CCONFIGPARAMS_H_
/**************************************************************************
****  DADF MODULE HEADER   ***
TYPE: Base Class |
PURPOSE: common access to user station configuration db
FUNCTION: Watch the configuration keys for changes and provide access to the different types of configuration parameter
INTERFACES: See ’INTERFACES:’ in the module declaration below
RESOURCES: --
REFERENCES: --
PROCESSING: --
DATA: See ’DATA:’ in the class header below.
LOGIC:

****  END MODULE HEADER   ***
**************************************************************************/
#include "BaseNT.h"
#include "string.h"
#include <vector>
#include "ErrorHandling.h"

#include "MSGTime.h"
#include "ConfigKeys.h"

namespace US
{
typedef struct
{
    DWORD min;
    DWORD value;
    DWORD max;
} SMinValueMax;
class CConfigParams
{
public:

    HANDLE m_ConfigChangeEvent;

    // initialize key depending variable
    CConfigParams();
    // Arguments: --
    // Returns: --
    // Exceptions: --

    ~CConfigParams();
    // Arguments: --
    // Returns: --
    // Exceptions: --

    // Initialize User station configuration root
    static LONG Initialize
    {
        // Arguments:
        // Path to the user station root
        const std::string& i_USConfigRoot;
        // Returns:
        // systemErrorCode ERROR_SUCCESS: completed successfully
        // other error occured
        // Exceptions: --

        // open User station relative key path
        LONG Open
        {
            // Arguments:
            // Setup relative key path
            const std::string& i_ParameterPath;
            // Returns:
            // systemErrorCode ERROR_SUCCESS: completed successfully
            // other error occured
            // Exceptions: --

            LONG OpenVolatile
            // Description: open User station relative key path - not saved
            {
                // Arguments:
                // Setup relative key path
                const std::string& i_ParameterPath;
                // Returns:
            }

    }

}
LONG DeleteKey();
// Description: delete the open key
// Arguments: --
// Returns: --
// Exceptions: --

HANDLE ChangeNotification();
// Arguments: --
// Returns:
// systemErrorCode ERROR_SUCCESS: completed successfully
// other error occured
// Exceptions: --

LONG Close();
// Arguments: --
// Returns:
// systemErrorCode ERROR_SUCCESS: completed successfully
// other error occured
// Exceptions: --

std::vector<std::string>& EnumSubKeys();

LONG Query(const std::string& i_ParameterName, std::string& o_ParameterValue);
// Returns:
// systemErrorCode ERROR_SUCCESS: completed successfully
// other error occured
// Exceptions: --

LONG CConfigParams::Query(const std::string& i_ParameterName, LPDWORD o_ParameterValue);
// Returns:
// systemErrorCode ERROR_SUCCESS: completed successfully
// other error occured
// Exceptions: --

// Set simple string parameter value
LONG SetSimpleString
{
    // Arguments:
    // Parameter name
    const std::string& i_ParameterName,
    // Parameter value
    std::string& i_ParameterValue
};

    // Returns:
    // systemErrorCode ERROR_SUCCESS: completed successfully
    // other error occured
    // Exceptions: --

// Set simple string parameter value
LONG SetDword
{
    // Arguments:
    // Parameter name
    const std::string& i_ParameterName,
    // Parameter value
    DWORD i_ParameterValue
};

    // Returns:
    // systemErrorCode ERROR_SUCCESS: completed successfully
    // other error occured
    // Exceptions: --

protected:

private:

    // DATA
    // Registry path to the configuration parameter root:
    // HKEY_LOCAL_MACHINE\Software\VCS\<US_ServiceName>\....
    // Service name depends on the installation
    static BOOL ms_Initialized; // false
    static std::string ms_ServiceKey;

    // Sub Path to the Facility
    std::string m_confSubKey;
    // Key pointing to the parameter
    HANDLE m_KeyHandle;

    // INTERFACE

    long ValueTypeError
    // Description: Log Value Type Error
    {
        // Arguments:
        // Parameter name
        const std::string& i_ParameterName,
        // Parameter type
        DWORD i_ParameterType,
        // Exspected parameter type
        DWORD i_ExspectedType
    };

    long ValueLengthError
    // Description: Log Value Length Error
    
const std::string& i_ParameterName,
    // Parameter length
   DWORD  i_ParameterLength,
    // max. space for parameter value
   DWORD  i_MaxSpace
);
// Return:  error status ERROR_INVALID_PARAMETER
// Exception: no specific

void ValueUnspecificError
// Description:  Log Value unspecific error
{
   // Arguments:
    const std::string& i_ParameterName,
    // system error status
    DWORD  status
);
// Return:  --
// Exception:  no specific

char *TypeMsg
// Description:  Return message text for registry value type
{
   // Arguments:
    registry value type
    DWORD  i_RegistryValueType
);
// Return:  message text
// Exception:  no specific

} // end namespace US
#endif

4.7.3 US::SKUcontrol

#ifdef _CSKUCONTROL_H
#define _CSKUCONTROL_H

/**************************************************************************
****  DADF MODULE HEADER   ***
TYPE: Concrete Class

PURPOSE: Handle access to the SKU

FUNCTION: Create a shared control memory and handle exclusively access to 
the SKU

*****************************************************************************/
#endif
INTERFACES: See 'INTERFACES:' in the module declaration below

RESOURCES: UTIL

REFERENCES: --

PROCESSING: Constructor creates a named mutex, which is used to control the exclusively access to the SKU. And connect its to the pbk shadow map, which contains the states of the sku pbk storage.

DATA: See 'DATA:' in the class header below.

LOGIC:

**** END MODULE HEADER ***
**************************************************************************
#include "MSGMutex.h"
#include "MSGTime.h"
#include "MSGEvent.h"
#include "GSDS_Volume_F.h"
#include "CUSLog.h"
#include "CConfigParams.h"
#include "CMemoryMap.h"

namespace US
{
    // Handling the PBK state map
    // --------------------------
    // state of pbk
    typedef enum
    {
        e_StateUnknown=0,
        e_StateSuccessfully = 1,
        e_StateFailed = 2
    } EpbkState;

    // struct of complete shadow
    struct SPBKShadow
    {
        char m_SKUStatusText[128];
        FILETIME m_update;
        struct
        {
            EpbkState m_state;
        }
    }
```cpp
FILETIME m_time;
    m_pbklist[256];
};

class CPBKShadow : public CMemoryMap
{
    public:
    // DATA:

    // INTERFACE:
    CPBKShadow ();
    // Description: constructor of this objects
    // Arguments: --
    // Returns: --
    // Exceptions: --

    ~CPBKShadow ();
    // Description: detstructor of this objects
    // Arguments: --
    // Returns: --
    // Exceptions: --

    bool Attach
    // Description: open the pbk shadow file and map to memory
    (
    // Arguments: --
    // File name of the pbk shadow
    std::string i_PBKshadowFileName,
    // Map name of the pbk shadow
    std::string i_PBKshadowMapName
    );
    // Returns: --
    // Exceptions:

    void Init ();
    // Description: init the pbk shadow
    // Arguments: --
    // Returns: --
    // Exceptions: --

    void SetGoodState
    // Description: set good state of key
    // Arguments:
    // Arguments:
    (
    // key no
    unsigned char i_keyNo,
    // attach a time
    Util::CMSGTime &i_pbkAttachedTime
    );
    // Returns: true, if save completes successfully
    // Exception: no specific

    void SetBadState
    // Description: save bad state of key
    // Arguments:
    //
    (
    // key no
    unsigned char i_keyNo,
    // attach a time
```
Util::CMSTime &i_pbkAttachedTime
);  // Returns: true, if save completes successfully
// false, if not
// Exception: no specific
void SetUnknownState
// Description: save bad state of key
// Arguments:
{
    key no
    unsigned char i_keyNo,
    // attach a time
    Util::CMSTime &i_pbkAttachedTime
};  // Returns: true, if save completes successfully
// false, if not
// Exception: no specific

void SetSKUStatus
// Description: set sku state
{
    char *i_SKUStatusText
};  // Returns: --
// Exceptions: no specific

EpbkState GetKeyState
// Description: get key state
// Arguments:
{
    key no
    unsigned char i_keyNo,
    // attached timestamp
    Util::CMSTime *o_timestamp
);
// Returns: state of the key defined by EpbkState
// Exception: no specific

char *GetSKUStatusText();
// Description: return sku status text
// Arguments: --
// Returns: pointer to status text
// Exceptions: no specific

private:
// DATA:
// entry for one public key
struct
{
    EpbkState m_state;
    FILETIME m_time;
} SPBKentry;

// File handling
HANDLE m_FileHandle;
// Data contained in the shadow
struct SPBKShadow *m_pbkMap;

//CSKUInquiry holds the SKU inquiry data
class CSKUInquiry
{
public:
    // DATA:
    bool m_valid;
    char m_FirmwareRevision[5]; // "0xFF"
    char m_HardwareRevision[5]; // "0xFF"
    char m_SKUNumber[7]; // "0xFFFF"
    char m_IntRomCheckSum[7]; // "0xFFFF"
    char m_StatusText[5]; // "0xFF"
    int m_StatusValue;

    // INTERFACE:
    CSKUInquiry();
    // Description: constructor
    // Arguments: --

    void Set
    // Description: Extract the relevant information from the SKU inquiry data
    // Arguments: const char i_skuinquiry[]
    // Returns: --
    // Exceptions: --

    void Reset
    // Description: reset the inquiry data
    // Arguments: --
    // Returns: --
    // Exceptions: --

    void Test
    // Description: set SKU test data
    // Arguments: --
    // Returns: --
    // Exceptions: --

    std::string Msg
    // Description: return Inquiry data as completed string
    // Arguments: --
    // Returns: --
    // Exceptions: --

protected:
    // repeat the above for the protected data/functions
private:

    // repeat the above for the private data/functions

};

// CSKUResult stores the result word returned by the sku response
class CSKUResult
{
    public:
        // DATA:
        unsigned int m_Result;
        bool m_valid;

        //INTERFACES:
        CSKUResult();
        // Description: Constructor
        // Returned: --
        // Exception: --

        void Set(char *i_result);
        // Description: convert result bytes to integer
        // Arguments: result bytes
        // Returned: --
        // Exception: --

        void Reset();
        // Description: reset the inquiry data
        // Arguments: --
        // Returns: --
        // Exceptions: --

        bool Bad();
        // Description: result indicates an error
        // Arguments: --
        // Returns: --
        // Exceptions: --

        void Test();
        // Description: set SKU test data
        // Arguments: --
        // Returns: --
        // Exceptions: --

        std::string Msg;
// Description: return Inquiry data as completed string
// ( // Arguments: -- ); // Returns: -- // Exceptions: -- ;

class CSKUControl
{
public:
  // DATA:
  // result of last sku operation (for sku access mmi)
  std::string m_ResultString;

  // INTERFACES:
  //
  CSKUControl(); // Description: constructor of this objects

  void Lock(); // Description: lock sku

  void Unlock(); // Description: unlock sku

  bool WritePublicKey( // Arguments:
    // key no
    unsigned char i_keyNo,
    // public key assigned to key no
    unsigned char i_publicKey[24],
    // crc
    unsigned short i_pbkCRC,
    // pbk attached time
    Util::CMSGTime i_pbkAttachedTime );
  // Returns: true, if successfully completed
  // false, if not
  // Exceptions: --

  bool ReadPublicKey
  (}
bool CalculatePNKey
// Description: read pseudo noise key
{
// Arguments:
//        key no  (read from the key header recode)
//        seed   (read from the key header record)
// pseudo noise key for PN pattern generation
// attached time to PBK

// Returns: true, if successfully completed
// false, if not

// Exceptions: --

bool CalculatePNKey
// Description: read pseudo noise key taking int parameter
{
// Arguments:
//        key no  (read from the key header recode)
//        seed   (read from the key header record)
// pseudo noise key for PN pattern generation

// Returns: true, if successfully completed
// false, if not

// Exceptions: --

bool Selftest
{
// Arguments:
// returns the result of the Selftest

// Returns: true, if successfully completed
// false, if not

// Exceptions: --
bool CSKUControl::Inquiry
// Description: read inquiry data
{
  // Arguments:
  // sku inquiry data
  class CSKUInquiry *o_InquiryData,
  // sku result
  class CSKUResult *o_Result
  );
  // Returns: true, if successfully completed
  // false, if not
  // Exceptions: Util::CParamException

US::EpbkState GetKeyState
// Description: get pbk state of the key
{
  // Argument:
  // key no
  unsigned char i_keyNo
  );
  // Returns: state of the key defined by EpbkState
  // Exception: no specific

HANDLE GetSignalEvent();
// Description: return the event set if something change
// in the pbk storage
// Arguments: --
// Returns: handle of event
// Exceptions: no specific

std::string GetSKUStatusText();
// Description: return the sku status text from pbk shadow
// Arguments: --
// Returns: string containing the text
// Exceptions: no specific

int GetSKUNumber() { return m_SKUNumber;};
// Description: return SKU No as hex value without '0x'
// Arguments: --
// Returns: sku no
// Exceptions: --

protected:

private:
// Interface:
bool US::CSKUControl::SKUCommand
// Description: send command string to sku and read response
{
  // Arguments:
  // sku command string as defined in SKU ICD
  const char *i_commandString,
  // sku response string
  char *o_response,
  // i: size of sku response and o: transferred sku response
  unsigned *t_responselength,
// pointer to the result infos in the response
char **o_result;

// Return: true, if command successfully completed and the response
// belongs to the command string
// false, else
// Exception: Util::CNamedException

// DATA:
// File associated with the shared memory
class CPBKShadow m_PBKshadow;

// Mutex to synchronize the SKU shadow access;
bool m_isLocked;
std::auto_ptr <class Util::CMMSGMutex> m_shadowLock;

// Event to signal pbk changes
std::auto_ptr <class Util::CMMSGEvent> m_shadowSignal;

// SKU number (if not connected -1)
int m_SKUNumber;

// end namespace US

#endif
## 5 Design Traces

### Design Traces Rev. 2.0

<table>
<thead>
<tr>
<th>US Ref. No.</th>
<th>Doc./Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>US.0020</td>
<td>[USDDS-A]; 3.1</td>
</tr>
<tr>
<td>US.0060</td>
<td>[USDDS-A]; 3.2.4.3</td>
</tr>
<tr>
<td>US.0070</td>
<td>[USDDS-A]; 3.2.4.3</td>
</tr>
<tr>
<td>US.0080</td>
<td>[USDDS-A]; 3.2.4.3</td>
</tr>
<tr>
<td>US.0090</td>
<td>not applicable for USDDS-A</td>
</tr>
<tr>
<td>US.0100</td>
<td>[USDDS-A]; 3.2.4.2</td>
</tr>
<tr>
<td>US.0110</td>
<td>[USDDS-A]; 3.1.1</td>
</tr>
<tr>
<td>US.0120</td>
<td>[USDDS-A]; 3.1.1</td>
</tr>
<tr>
<td>US.0150</td>
<td>[USDDS-A]; sec. 7</td>
</tr>
<tr>
<td>US.0330</td>
<td>[USDDS-A]; 3.2.4.1-3.2.4.2</td>
</tr>
<tr>
<td>US.0560</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0570</td>
<td>[USDDS-A]; 3.2.4.1+7.6.1.6</td>
</tr>
<tr>
<td>US.0580</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0590</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0600</td>
<td>[USDDS-A]; 3.2.4.1+3.2.6.2</td>
</tr>
<tr>
<td>US.0610</td>
<td>[USDDS-A]; 3.2.4.1+7.4</td>
</tr>
<tr>
<td>US.0620</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0630</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0640</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0650</td>
<td>[USDDS-A]; 3.2.4.1+7.6.1.6</td>
</tr>
<tr>
<td>US.0660</td>
<td>[USDDS-A]; 3.2.4.1+7.6.1.6</td>
</tr>
<tr>
<td>US.0670</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0680</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0690</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.0700</td>
<td>[USDDS-A]; 7.6.1.4+3.2.7.1</td>
</tr>
<tr>
<td>US.0710</td>
<td>[USDDS-A]; 3.2.4.1+3.2.7.1</td>
</tr>
<tr>
<td>US.0720</td>
<td>not applicable for USDDS-A</td>
</tr>
<tr>
<td>US.0730</td>
<td>[USDDS-A]; 3.2.7.1</td>
</tr>
<tr>
<td>US.0740</td>
<td>[USDDS-A]; 7.8</td>
</tr>
<tr>
<td>US.0750</td>
<td>[USDDS-A]; 7.8</td>
</tr>
<tr>
<td>US.0760</td>
<td>[USDDS-A]; 7.5.3</td>
</tr>
</tbody>
</table>
US.0780 [USDDS-A]; 3.2.4.2
US.0790 [USDDS-A]; 3.2.4.2
US.0800 [USDDS-A]; 3.2.2.1
US.0810 [USDDS-A]; 3.2.4.2 + 3.2.6.2
US.0840 [USDDS-A]; 3.2.4.2
US.0850 [USDDS-A]; 3.2.4.2
US.0860 [USDDS-A]; 3.2.4.2 + 3.2.6.2
US.0870 [USDDS-A]; 7.6.2
US.0880 [USDDS-A]; 7.6.2
US.0890 [USDDS-A]; 3.2.4.4 + 7.8
US.0920 [USDDS-A]; 3.2.4.2
US.0930 [USDDS-A]; 3.2.4.2
US.0940 [USDDS-A]; 3.2.4.2
US.0950 [USDDS-A]; 3.2.4.2
US.0960 [USDDS-A]; 3.2.4.2 + 3.2.6.2
US.0970 [USDDS-A]; 7.8
US.0990 [USDDS-A]; 3.2.4.2 + 3.2.4.3
US.1000 [USDDS-A]; 7.4
US.1010 [USDDS-A]; 7.5.1
US.1020 [USDDS-A]; 3.2.4.2
US.1030 [USDDS-A]; 3.2.4.2
US.1040 [USDDS-A]; 3.2.4.2 + 7.5.1+7.5.2
US.1050 [USDDS-A]; 7.5.1
US.1060 [USDDS-A]; 7.5.1+7.5.2
US.1070 [USDDS-A]; 3.2.4.2 + 7.5.1+7.5.2
US.1080 [USDDS-A]; 3.2.4.2 + 7.5.1+7.5.2
US.1090 [USDDS-A]; 7.8 + 7.5.3
US.1100 [USDDS-A]; 3.2.4.3
US.1110 [USDDS-A]; 3.2.4.3
US.1111 [USDDS-A]; 3.2.4.2 + 3.2.4.3
US.1120 [USDDS-A]; 3.2.4.2 + 3.2.4.3
US.1130 [USDDS-A]; 3.2.4.2
US.1131 [USDDS-A]; 3.2.4.2
US.1135 [USDDS-A]; 3.2.4.2 + 7.6.3
US.1140 [USDDS-A]; 7.6.1.2
US.1160 [USDDS-A]; 3.2.4.1 + 3.2.4.3
US.1170 [USDDS-A]; 3.2.4.3
### Table 5-1: Design Traces

<table>
<thead>
<tr>
<th>US.1180</th>
<th>[USDDS-A]; 3.2.4.3 + 3.2.6.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>US.1190</td>
<td>[USDDS-A]; 3.2.4.3</td>
</tr>
<tr>
<td>US.1200</td>
<td>[USDDS-A]; 3.2.4.3</td>
</tr>
<tr>
<td>US.1210</td>
<td>[USDDS-A]; 7.8</td>
</tr>
<tr>
<td>US.1220</td>
<td>[USDDS-A]; 7.4 + 3.2.6.2</td>
</tr>
<tr>
<td>US.1230</td>
<td>[USDDS-A]; 3.2.6.2</td>
</tr>
<tr>
<td>US.1240</td>
<td>[USDDS-A]; 3.2.4.1</td>
</tr>
<tr>
<td>US.1250</td>
<td>[USDDS-A]; sec. 7</td>
</tr>
<tr>
<td>US.1270</td>
<td>[USDDS-A]; 3.2.4.1 + 7.4</td>
</tr>
<tr>
<td>US.1280</td>
<td>[USDDS-A]; 3.2.4.1 + 7.4</td>
</tr>
<tr>
<td>US.1290</td>
<td>[USDDS-A]; 3.2.4.1 + 7.4</td>
</tr>
<tr>
<td>US.1300</td>
<td>[USDDS-A]; 7.4</td>
</tr>
<tr>
<td>US.1310</td>
<td>[USDDS-A]; 7.4</td>
</tr>
<tr>
<td>US.1320</td>
<td>[USDDS-A]; 7.4</td>
</tr>
<tr>
<td>US.1330</td>
<td>[USDDS-A]; 3.2.4.1 - 3.2.4.3 + 3.2.6.2</td>
</tr>
</tbody>
</table>
6 Glossary
7 Annex A: User Station MMIs

7.1 Introduction

7.1.1 Purposes
The purpose of this annex is to define the MMI design of the User station screens. The guideline of the design is to fulfill the specific user station MMI requirements but also to maximize reuse of MMIs designed for the DADF facility. MMIs used for the User Station and for a DADF facility are referenced in this document but not specified.

7.1.2 MMI Tools and Environments
The list of tools used for MMI production is given below.

- Visual C++
  Used as the general MMI development environment.

- Windows NT
  Used for most of the systems screens, and some generic screens, print dialog, font selection etc.
  More detail is given in the [FMPS].

- Visual Source Safe
  Used for configuration control within the DADF.

- Windows NT Utilities

7.1.3 Story Boards
The initial stage of MMI development is to produce a series of storyboards, identifying the screens required and showing the hierarchy of screen navigation for the whole MMI. They provide a representation of the main underlying functionality of the facility screens.

These screen storyboards are included in the relevant sections and use the following notation:

- Square boxes represent menu buttons.
- Solid line ovals represent screens.
- Dotted line ovals represent other individual screens that are called from the central screen.
- Single arrowhead lines represent a simple call, such as opening a new screen.
- Double arrowheads represent some more complex underlying functionality, which may be called by the user (e.g. pressing a button to toggle gridlines off), or which may be called by the system (e.g. updating the parameter value field in response to incoming data).
7.1.4 Hierarchy diagram

<table>
<thead>
<tr>
<th>User Station banner</th>
<th>File</th>
<th>File Manager (NT Explorer)</th>
<th>File viewer ASCII&amp;HEX</th>
<th>Historical Image viewer</th>
<th>Trace Packet Viewer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitoring</td>
<td>Data Stream Monitor</td>
<td>SKU Status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Realtime Image Viewer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>Configuration Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Station Key Unit access</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Edit Transfer Filter</td>
<td>Edit Filter Entry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log</td>
<td>Realtime Logview</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Historical Logview</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Help</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.2 Overview

The user station MMIs fall into the following groups.

**User Station banner**

The banner allows access to all the screens available for the User Station. All screens launched from the banner will reference files that are associated with the User Station type (HRIT or LRIT) shown in the banner.

**Monitoring screens**

The monitoring screens display the current state of the User Station system:

The data stream monitoring display.
Viewers

These are the main views for monitoring the data reception and processing and for displaying historical data based on files. These are for

Monitoring:
- Image view for Level 1.5 images, MPEF products and foreign satellite images

Historical Data:
- The trace packet view for VCDU or Source Packets
- Generic file view for LRIT/HRIT files
- Image view for Level 1.5 images, MPEF products and foreign satellite images

See Analysis of DADF-MMIs for more details

Control

They are composed of:
- The configuration display
- The SKU access display
- The transfer filter editor

Log

Realtime logview
Historical Logview

Both viewers are provided by the VCS Log display.

File Manager

Adapted NT Explorer
7.3 User Station banner

Description: Allows access to User Station screens.

Additional comments:

Story Board
Window elements of User Station  Banner

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>General</td>
<td>Screen requirement</td>
</tr>
<tr>
<td></td>
<td>requirement</td>
<td></td>
</tr>
<tr>
<td>Toolbar</td>
<td>Toolbars</td>
<td>Restore, move, size x, size y, minimise, maximise, close</td>
</tr>
<tr>
<td>Menu</td>
<td>Menu</td>
<td>See hierarchy section.</td>
</tr>
</tbody>
</table>

Behaviour

Requirements:

7.4  Data Stream Monitor screens

Picture
### Baseband Status

| Timestamp          | Demod | BSinc | FSyncEn | FSync | VONSync | InpAGC (%) | InpDiff (kHz) | Eb/No (dB) | VSTBER  | VLBER   | XSTPFL | XLTPFL |
|--------------------|-------|-------|---------|-------|---------|------------|---------------|------------|----------|---------|--------|--------|--------|
| 1999/08/13 14:36:45.174 | Unlocked | Locked | 12      | Unlocked | Locked | 3          | 1             | 0          | 0.00e+000 | 1.00e+000 | 0.00e+000 | 0.00e+000 |
| 1999/08/13 14:36:35.428 | Unlocked | Locked | 12      | Unlocked | Locked | 3          | 1             | 0          | 0.00e+000 | 1.00e+000 | 0.00e+000 | 0.00e+000 |
| 1999/08/13 14:36:12.786 | Unlocked | Locked | 12      | Unlocked | Locked | 3          | 1             | 0          | 0.00e+000 | 0.00e+000 | 0.00e+000 | 0.00e+000 |
| 1999/08/13 14:36:02.621 | Unlocked | Locked | 34      | Unlocked | Locked | 3          | 54            | 0          | 0.00e+000 | 0.00e+000 | 0.00e+000 | 0.00e+000 |
| 1999/08/13 14:35:36.052 | Unlocked | Locked | 34      | Unlocked | Locked | 3          | 57            | 0          | 0.00e+000 | 0.00e+000 | 0.00e+000 | 0.00e+000 |
| 1999/08/13 14:35:45.607 | Unlocked | Locked | 0       | Unlocked | Locked | 3          | 57            | 0          | 0.00e+000 | 0.00e+000 | 0.00e+000 | 0.00e+000 |
| 1999/08/13 14:35:40.609 | Unlocked | Locked | 0       | Unlocked | Locked | 3          | 0             | 0          | 0.00e+000 | 0.00e+000 | 0.00e+000 | 0.00e+000 |

### Data Monitor

#### VCDU Reception

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Filler (%)</th>
<th>BER (%)</th>
<th>Correct (%)</th>
<th>Corrupt (%)</th>
<th>Decrypt (%)</th>
<th>Decompress (%)</th>
<th>Monitor (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999/08/13 14:36:34.561</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
</tr>
<tr>
<td>1999/08/13 14:36:34.561</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
</tr>
<tr>
<td>1999/08/13 14:36:34.561</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
</tr>
<tr>
<td>1999/08/13 14:36:34.561</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
</tr>
<tr>
<td>1999/08/13 14:36:34.561</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
</tr>
<tr>
<td>1999/08/13 14:36:34.561</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
</tr>
<tr>
<td>1999/08/13 14:36:34.561</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
</tr>
<tr>
<td>Accumulated</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
<td>0.00e+000</td>
</tr>
</tbody>
</table>

#### XRT Files

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Monitor (%)</th>
<th>SKY (Status)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKY</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

#### DISE Transfer

- Monitoring: [Red Bar]
- File Data: [Red Bar]
Description: This MMI shows the actual status information about the data reception and processing.

Additional comments:

## Window elements of Data Stream Monitor window

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>General requirement</td>
<td>Screen General Screen requirement</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Indication text</td>
<td>Shows the time the baseband information are received</td>
</tr>
<tr>
<td>(Baseband Status)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DemLock</td>
<td>Indication text</td>
<td>Shows if the demodulator is locked or not</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSyncLock</td>
<td>Indication text</td>
<td>Shows if the bit synchronizer is locked or not</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FSyncErr</td>
<td>Indication text</td>
<td>Shows the number of errors in the frame sync mark pattern.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FSyncLock</td>
<td>Indication text</td>
<td>Shows if the frame synchronizer is locked or not</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDNSync</td>
<td>Indication text</td>
<td>Shows if the Viterbi decoder node is locked or not</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InpAGC</td>
<td>Indication text</td>
<td>Shows the current AGC level for the input signal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InpOffset</td>
<td>Indication text</td>
<td>Shows the current frequency offset for the input signal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eb/No</td>
<td>Indication text</td>
<td>Shows the Eb/No estimate for the input signal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VSTBER</td>
<td>Indication text</td>
<td>Shows the short-term bit error rate derived from the Viterbi decoder</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLTBER</td>
<td>Indication text</td>
<td>Shows the long-term bit error rate derived from the Viterbi decoder</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSTPFL</td>
<td>Indication text</td>
<td>Shows the short-term probability of frame loss derived from the RS decoder</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XLTPFL</td>
<td>Indication text</td>
<td>Shows the long-term probability of frame loss derived from the RS decoder</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td>Indication text</td>
<td>Shows the time the end of the count interval is reached.</td>
</tr>
<tr>
<td>(Data Monitor)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### NAME | ITEM | ITEM BEHAVIOUR
--- | --- | ---
Filler | Indication text | Shows the percent of filler VCDUs from all received VCDUs in the time interval.
BER (VCDU Reception) | Indication text | Shows the percent of bit errors from all bits in the received filler VCDUs in the time interval.
Corrected (VCDU Reception) | Indication text | Shows the percent of corrupted VCDUs, which were corrected in the time interval. Base are all received VCDUs.
Corrupted (VCDU Reception) | Indication text | Shows the percent of corrupted VCDUs, which were not corrected in the time interval. Base are all received VCDUs.
Corrupted (XRIT Files) | Indication text | Shows the percent of corrupted files from all received XRIT files in the time interval.
Decrypted (XRIT Files) | Indication text | Shows the percent of successfully decrypted files from all encrypted files in the time interval.
Decompressed (XRIT Files) | Indication text | Shows the percent of successfully decompressed files from all compressed files in the time interval.
Monitoring (DISE Transfer) | Indication text | Shows the percent of transferred monitoring data from available monitoring data in the time interval.
XRITE Files (DISE Transfer) | Indication text | Shows the number of files send to DISE in the time interval.
SKU | Button and indication text | Opens the SKU status window. The second line of the button text displays the current SKU status.
DISE Link Status | Indication lamp | Shows if DISE is connected to receive Monitoring or/and File Data.

### Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseband link status</td>
<td>Shows the base band status received from the MUBM. The base band status is routinely received from the MUBM. A new line is added to the top of the list, if a status value changes significantly.</td>
</tr>
</tbody>
</table>
### NAME | ITEM BEHAVIOUR
---|---
Data Monitor | Shows statistical information about the data processing. Each line in the list represents the accumulation of the shown values for a short time interval. The first line after the list, titled ‘Accumulated’ shows the sum of all values of the same type in the list (long time interval).

**Requirements**

- US.0020
- US.0610
- US.1000
- US.1270
- US.1290
- US.1300
- US.1310
- US.1320

#### 7.4.1 SKU Status

**Description:** This window shows the current state of the PBK storage and the last known state of the SKU device.

**Additional comments:**

![SKU Status Diagram](image-url)
Window elements of SKU Status window

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKU</td>
<td>Indication text</td>
<td>Shows the last known status of the SKU device</td>
</tr>
<tr>
<td>PBK Storage</td>
<td>Grid</td>
<td>Shows the state of PBKs for each possible key. The following states are possible:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘-’ : not set (neither by last received EKM nor by the SKU access MMI)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘S’: Successfully set to PBK storage in the SKU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘E’: Error occurred if set new PBK (CRC or communication error)</td>
</tr>
</tbody>
</table>

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBK Storage</td>
<td>The state of the PBK storage is updated each time a single PBK or the content of a complete EKM file shall be loaded to the SKU device. The state ‘not set’ is entered, if last received EKM file contains no PBK for that key. The state ‘S’ is entered if a PBK is loaded successfully to the SKU device and the state ‘E’ is entered, if wether the SKU detects a CRC error for that PBK or the command transfer to the SKU failed.</td>
</tr>
</tbody>
</table>

Requirements : US.1300

7.5 Viewers

7.5.1 Realtime Image Viewer

The realtime image viewer display the available data of an image file currently received. The displayed data are subsampled to fit into a defined MMI size. The content of the window is automatically updated if new data available.
Description: Shows currently available data of an image product.

Additional comments:
Produced using: Visual C++

Window elements of Level 1.5 Image view (SEVIRI Image Monitoring)
### Annex A: User Station MMIs

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Screen</td>
<td>requirement</td>
<td>General Screen requirement</td>
</tr>
<tr>
<td>Channel/Product</td>
<td>Selection list</td>
<td>Shows a list of available channels or products.</td>
</tr>
<tr>
<td>Image area</td>
<td>Image display</td>
<td>This shows the available part of a subsampled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>image. Corrupted or Lost data are marked coloured.</td>
</tr>
<tr>
<td>Time</td>
<td>Indication text</td>
<td>Shows the acquisition or nominal product time of that image.</td>
</tr>
<tr>
<td>Origin</td>
<td>Selection list</td>
<td>Shows a list image data sources, which are available</td>
</tr>
<tr>
<td>Dissemination S/C</td>
<td>Indication text</td>
<td>Shows the satellite id which has disseminated this image.</td>
</tr>
<tr>
<td>Reception Starttime</td>
<td>Indication text</td>
<td>This timestamp marks the reception of the first data package for the XRITE file containing this image</td>
</tr>
<tr>
<td>Data lost</td>
<td>Indication text</td>
<td>Shows how much of the image data are not received.</td>
</tr>
<tr>
<td>Data corrupted</td>
<td>Indication text</td>
<td>Shows how much of the image data are received, but corrupted</td>
</tr>
<tr>
<td>File-Exit</td>
<td>Menu</td>
<td>Exit realtime image view</td>
</tr>
<tr>
<td>View</td>
<td>Menu</td>
<td></td>
</tr>
</tbody>
</table>

### Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAGE HANDLING</td>
<td>After image data of a received image file is subsampled, these data are passed through a generic server, which stores the data for the MMI. If a user selects Origin and a channel respectively product, the MMI request the data from that generic server, which stores the selected image data. As soon as the reception of an image of the same type starts, the existing old one is overwritten within the same generic server. There is one controlling generic server, which holds a list of images available for the realtime image viewer.</td>
</tr>
</tbody>
</table>
Requirements:

- US.0020
- US.1010
- US.1020
- US.1040
- US.1050
- US.1060
- US.1070
- US.1080

### 7.5.2 Historical Image Viewer

The historical image-viewer displays the contents of a subsampled image file.

Description: Shows a complete subsampled image

Additional comments:

Produced using: Visual C++

Window elements of Level 1.5 Image view (SEVIRI Image Monitoring)
### NAME | ITEM | ITEM BEHAVIOUR
---|---|---
| General Screen requirement | Screen | General Screen requirement |
| Time controls | Time controls | Select one repeat cycle |
| Channel/Product | Selection list | Shows a list of available channels or products |
| Image area | Image display | This shows a subsampled image |
| Origin | Selection list | Shows a list image data sources, which are available |
| Time | IndicationText | Reception Time |
| File-Open | Menu | Select a subsampled image file for display by the standard open dialog |
| File-Exit | Menu | Exit realtime image view |
| View | Menu | |

### Behaviour

#### NAME | ITEM BEHAVIOUR
---|---
| IMAGE HANDLING | If a user has selected an image, the MMI opens and loads the subsampled file to the image area. The subsampled file contains a list of image lines. Each line has the same structure as an image line for the realtime image viewer. |

### Requirements:

US.0020US.1010US.1020
US.1030US.1040US.1050
US.1060US.1070US.1080

#### 7.5.3 File viewer – ASCII&HEX

The contents of a received and/or processed L/HRIT file is displayed in a dump view.

Picture
For illustrative purposes only.

Description: Allows the dump view of XRIT files in hex and ASCII. The number of items shown on a row can be changed by dragging the table header of the window left right. The size of the ASCII window is directly proportioned to the hex window. A line can be reversed by selecting the line and choosing reverse from the pop up menu.
Additional comments
Produced using: Visual C++

Story Board
Generic Viewer
- ASCII / HEX

- File: Open
- File: Exit
- Popu menu: reverse line
- Help

Generic File
Open display

move split
left right
change number of
values per line
Window elements of File viewer

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reverse line</td>
<td>Popup menu</td>
<td>This menu allows a particular line to be reversed. This will be removed if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>number of values per line is changed. Any number of lines can be reversed.</td>
</tr>
<tr>
<td>Values per line</td>
<td>List control</td>
<td>Change the number of values per line by moving the splitter left right.</td>
</tr>
<tr>
<td></td>
<td>title</td>
<td></td>
</tr>
<tr>
<td>Hex</td>
<td>Within list</td>
<td>This shows the hex values. IT shows a double value FF and then a space so a</td>
</tr>
<tr>
<td></td>
<td>control</td>
<td>typical line will be “FF EF 14 FE”. The number of values shown per line can</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be changed by altering the size of the table.</td>
</tr>
<tr>
<td>ASCII</td>
<td>Within list</td>
<td>This shows the values in ASCII format. This will be in the format abc12$...$</td>
</tr>
<tr>
<td></td>
<td>control</td>
<td>where full stops are used for values which can’t be displayed.</td>
</tr>
<tr>
<td>Location in file</td>
<td>Within list</td>
<td>This is another column which may or may not be shown on the normal screen</td>
</tr>
<tr>
<td></td>
<td>control</td>
<td>area depending on the number of hex values per line selected. If it is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>shown on the screen scroll bars can be used to reach this column. The</td>
</tr>
<tr>
<td></td>
<td></td>
<td>column will have two values one for the length of the first value of the item</td>
</tr>
<tr>
<td></td>
<td></td>
<td>from the start of the file in bytes and the other for the last value on the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>line.</td>
</tr>
<tr>
<td>Location in file</td>
<td>Scroll bar</td>
<td>The scroll bar will allow the user to</td>
</tr>
</tbody>
</table>

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
</table>

Requirements : US.0760  US.1090

7.5.4 Trace Packet Viewer
7.5.4.1 VCDU Trace Viewer

Picture
Description: This window displays the contents of a trace buffer, containing VCDU data units, received from the MUBM.

Windows: see picture

Additional comments: The trace buffer is created during the VCDU trace mode for the File Assembler (see section 3.2.4.1) is switched on. During trace mode all received VCDU are stored to a cyclic buffer, called
trace buffer. After trace mode is switched off, the trace buffer is saved to disk. Files of this structure can be displayed by this viewer.

Window elements of VCDU Trace Viewer

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Number</td>
<td>NumberControl</td>
<td>Shows the currently displayed vcdu packet within the vcdu trace buffer and it is used to go across the buffer.</td>
</tr>
<tr>
<td>Sync marker</td>
<td>Indication text</td>
<td>Shows the 32-bit sync marker value attached to this vcdu</td>
</tr>
<tr>
<td>VCDU area</td>
<td>View area</td>
<td>Shows the contents of the selected VCDU in HEX and ASCII interpretation.</td>
</tr>
<tr>
<td>Reed Solomon area</td>
<td>View area</td>
<td>Shows the reed solomon check symbols in HEX and ASCII interpretation.</td>
</tr>
<tr>
<td>Input AGC</td>
<td>Indication field</td>
<td>Shows the input AGC</td>
</tr>
<tr>
<td>MUBM Timestamp</td>
<td>Indication filed</td>
<td>Shows the MUBM timer value attached to the VCDU</td>
</tr>
<tr>
<td>RS error flag 0 … 3</td>
<td>Indication box</td>
<td>Indicate if there are uncorrectable reed solomon errors in the VCDU</td>
</tr>
<tr>
<td>No frame synconiser lock</td>
<td>Indication box</td>
<td>Indicates frame synconiser lock or not</td>
</tr>
<tr>
<td>Flywheel</td>
<td>Indication box</td>
<td>Indicates fly wheel synconisation if no frame sync lock</td>
</tr>
<tr>
<td>Inverted Syncmarker</td>
<td>Indication box</td>
<td>Inverted sync pattern found</td>
</tr>
<tr>
<td>Details</td>
<td>Button</td>
<td>Opens an additional window, displays the expanded quality information</td>
</tr>
<tr>
<td>File-Open</td>
<td>Menu</td>
<td>Opens a trace buffer file</td>
</tr>
<tr>
<td>File-Exit</td>
<td>Menu</td>
<td>Exit the viewer</td>
</tr>
</tbody>
</table>

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation</td>
<td>The time control is used to go across the trace buffer.</td>
</tr>
<tr>
<td>View area</td>
<td>Scroll up and down to view the complete area</td>
</tr>
</tbody>
</table>
Requirements: --

7.5.4.2 VCDU Trace Details

Description: This window displays the quality information attached to the MUBM VCDU data package. Windows: see picture
Additional comments: The trace buffer is created during the VCDU trace mode for the File Assembler (see section 3.2.4.1) is switched on. During trace mode all received VCDU are stored to a cyclic buffer, called trace buffer. After trace mode is switched off, the trace buffer is saved to disk.

Window elements of VCDU Quality details

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>General Screen</td>
<td>General Screen requirement</td>
</tr>
<tr>
<td>Value fields</td>
<td>Indication fields</td>
<td>Shows the value of the specified parameter attached to the displayed VCDU. See [US Detailed design specification – ICD: MUBM::Workstation Issue 2.0; section 3.2 Data definitions] for a description of the different fields</td>
</tr>
<tr>
<td>Flag fields</td>
<td>Indication box</td>
<td>Shows the state of a flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓ : flag set or empty: flag reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See [US Detailed design specification – ICD: MUBM::Workstation Issue 2.0; section 3.2 Data definitions] for a description of the different fields</td>
</tr>
<tr>
<td>Exit</td>
<td>Button</td>
<td>Exit the viewer</td>
</tr>
</tbody>
</table>

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents of fields</td>
<td>This window shows the quality information attached to the VCDU, displayed in the VCDU trace viewer. If another VCDU data package is selected for display, the contents of the VCDU Quality Details is also changed.</td>
</tr>
<tr>
<td>Close window</td>
<td>Close the window</td>
</tr>
</tbody>
</table>

Requirements: --

7.5.4.3 Source Packet Trace Viewer

Picture
Description: This window displays the contents of a trace buffer.

Windows: see picture

Additional comments: The trace buffer is created during the source packet trace mode for the File Assembler (see section 3.2.4.1) is switched on. During trace mode all assembled source packets are stored to a cyclic buffer, called trace buffer. After trace mode is switched off, the trace buffer is saved to disk. Files of this structure can be displayed by this viewer.
Window elements of Source Packet Trace Viewer

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Number</td>
<td>Number control</td>
<td>Shows the packet number of the VCDU belongs to the displayed packet in the view area and is used to go across the buffer.</td>
</tr>
<tr>
<td>Source packet area</td>
<td>View area</td>
<td>Shows the contents of the selected source packet in HEX and ASCII interpretation.</td>
</tr>
<tr>
<td>CRC</td>
<td>Indication filed</td>
<td>Shows the result of the source packet CRC check</td>
</tr>
<tr>
<td>CRC Check sum</td>
<td>Indication field</td>
<td>Shows the source packet data field check sum</td>
</tr>
<tr>
<td>Header</td>
<td>Indication field</td>
<td>Shows the source packet header</td>
</tr>
<tr>
<td>Corrupted data</td>
<td>Indication text</td>
<td>Shows the count of the corrupted data within the source packet data field</td>
</tr>
<tr>
<td>File-Open</td>
<td>Menu</td>
<td>Opens a trace buffer file</td>
</tr>
<tr>
<td>File-Exit</td>
<td>Menu</td>
<td>Exit the viewer</td>
</tr>
</tbody>
</table>

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation</td>
<td>The time control is used to go across the trace buffer.</td>
</tr>
<tr>
<td>View area</td>
<td>Using scrollbars to navigate in the packet contents</td>
</tr>
</tbody>
</table>

Requirements: --

7.6 Control

7.6.1 User Station Configuration Control
Description: The configuration control provides the user interface to the configuration parameter stored in the NT registry. The parameters are subdivided into tabs.

Additional comments:

Window elements of Configuration Control

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabs</td>
<td>Tab</td>
<td>Grouping of the different parameter types</td>
</tr>
<tr>
<td>Ok</td>
<td>Button</td>
<td>Save the changeable parameter of the selected tab to the NT registry and close the window</td>
</tr>
<tr>
<td>Apply</td>
<td>Button</td>
<td>Save the changeable parameter of the selected tab to the NT registry.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Button</td>
<td>Close the window.</td>
</tr>
<tr>
<td>File-exit</td>
<td>Menu</td>
<td>Same as cancel button</td>
</tr>
</tbody>
</table>
## Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Requirements ::

7.6.1.1 General tab

### Picture

![User Station Configuration Control](image-url)
Description: This tab is part of the User station configuration control window and shows some general information about the MSG User station.

Additional comments: The displayed revisions and the SKU No are hard coded information. They do not change until an hard – or software update is done.

Window elements of General tab

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Station</td>
<td>Indication text</td>
<td>Shows the type of User station: HRIT or LRIT</td>
</tr>
<tr>
<td>Software revision</td>
<td>Indication text</td>
<td>Shows the version of the installed user station software.</td>
</tr>
<tr>
<td>MUBM revision</td>
<td>Indication text</td>
<td>Shows the version of the connected MUBM device.</td>
</tr>
<tr>
<td>SKU No</td>
<td>Indication text</td>
<td>Shows the Number of the connected Station Key Unit.</td>
</tr>
<tr>
<td>Ok</td>
<td>Button</td>
<td>Close the configuration control window</td>
</tr>
<tr>
<td>Apply</td>
<td>Button</td>
<td>Disabled (greyed), this tab is only for informational purposes</td>
</tr>
<tr>
<td>Cancel</td>
<td>Button</td>
<td>Close the configuration control</td>
</tr>
</tbody>
</table>

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Requirements

::
7.6.1.2 Network tab

Description: This tab is part of the User station configuration control window and shows the configuration parameter for the DISE transfer.

Additional comments: If the Port Ids are not set, the user station is configured as a standalone system.

Window elements of Network tab

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port Id</td>
<td>Indication text</td>
<td>This field shows the Port Id used to transfer the monitoring data to DISE</td>
</tr>
<tr>
<td>(Monitoring Data..)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port Id</td>
<td>Indication text</td>
<td>This field shows the Port Id used to transfer the file data to DISE</td>
</tr>
</tbody>
</table>
(File Data ...)

<table>
<thead>
<tr>
<th>Disable Buffer</th>
<th>Check box</th>
<th>Disable buffering of monitoring data if DISE link is broken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted File</td>
<td>Combo box</td>
<td>This will show the interval between encrypted files send to DISE. Ranges are from every 100 to every 1000 file to be transferred. The ranges are shown in the combo box list.</td>
</tr>
<tr>
<td>Compressed File</td>
<td>Combo box</td>
<td>See Encrypted File, but for compressed files</td>
</tr>
<tr>
<td>Decompressed File</td>
<td>Combo box</td>
<td>See Encrypted File, but for decompressed files</td>
</tr>
<tr>
<td>Ok</td>
<td>Button</td>
<td>Save the parameter of this tab to NT registry and close the window</td>
</tr>
<tr>
<td>Apply</td>
<td>Button</td>
<td>Save the parameter of this tab to the NT registry.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Button</td>
<td>Undo parameter modification of this tab.</td>
</tr>
</tbody>
</table>

**Behaviour**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Requirements ::** US.0080 US.1120 US.1140

7.6.1.3 Automatic Transfer

Picture
Description: This tab is part of the User station configuration control window and defines the filter and destination root for the automatic file transfer to a location outside the online buffer. If the FTP Transfer is configured, the files are transferred using the FTP service.

Additional comments: If no filter is specified, no files are transferred.

Window elements of Automatic Transfer tab

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer Root</td>
<td>Indication and edit text</td>
<td>Shows the root destination of the automatic file transfer.</td>
</tr>
<tr>
<td>Transfer Filter</td>
<td>Indication and edit text</td>
<td>Shows the Filter file used to select files for automatic transfer.</td>
</tr>
<tr>
<td>Remote computer</td>
<td>Edit text</td>
<td>Specify the remote computer for the FTP transfer</td>
</tr>
<tr>
<td>User name</td>
<td>Edit text</td>
<td>Specify a user to of the remote computer.</td>
</tr>
<tr>
<td>Password</td>
<td>Password text</td>
<td>Specify a password for the user</td>
</tr>
<tr>
<td>Ok</td>
<td>Button</td>
<td>Save the parameter of this tab to NT registry and close the window</td>
</tr>
</tbody>
</table>
**Apply** Button Save the parameter of this tab to the NT registry.
**Cancel** Button Undo parameter modification of this tab.

### Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Requirements:: US.1130
7.6.1.4 Buffer Sizes

Description: This tab is part of the User station configuration control window and defines the buffer sizes. Each one of the buffer has to store data for a configurable time.

Additional comments: Modification of these parameter activated after a restart of the user station software.

Window elements of Buffer Sizes tab

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>General</td>
<td>Screen General Screen requirement</td>
</tr>
<tr>
<td>… Trace …</td>
<td>Text Box</td>
<td>The first box shows the storage time for a trace packet in the trace buffer. The second and the third boxes defines the range (min/max) for the trace data storage time.</td>
</tr>
<tr>
<td>… Quicklook …</td>
<td>Text box</td>
<td>The first box shows the storage time for the</td>
</tr>
</tbody>
</table>
subsampled image files after the last modification. The second and the third boxes defines the range (min/max) for the storage time.

The first box shows the storage time for the received and processed L/HRIT files. The second and the third boxes defines the range (min/max) for the storage time.

Save the parameter of this tab to NT registry and close the window

Save the parameter of this tab to the NT registry.

Undo parameter modification of this tab.

Requirements: US.1030
7.6.1.5 Base Band

Description: This tab is part of the User station configuration control window and defines parameters used by the MUBM during data preprocessing.

Additional comments: After parameters are stored to the NT registry, they are passed to the MUBM via the SCSI connection.

**Window elements of Base Band tab**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. Allowed</td>
<td>Number</td>
<td>Allowed number of bit (±) slips of the frame synchroniser.</td>
</tr>
<tr>
<td>of Bit Slips</td>
<td>Indication and edit text</td>
<td>The first box shows the current parameter value and is editable. The second</td>
</tr>
<tr>
<td>Max. Permitted</td>
<td></td>
<td>and the third boxes</td>
</tr>
<tr>
<td>Number of Bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock Threshold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unlock Threshold</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
defines the range (min/max) for this parameter.

Max. permitted number of bit errors  
Indication and edit text  
Allowed number of bit errors in the frame synchronisation marker.

The first box shows the current parameter value and is editable. The second and the third boxes defines the range (min/max) for this parameter.

Pattern length  
Indication text  
Length of synchronisation marker.

Unlock Threshold  
Indication and edit text  
Fly wheel parameter of the frame synchroniser. Number of unfound fsync markers before ‘frame sync lock loss’ is detected.

The first box shows the current parameter value and is editable. The second and the third boxes defines the range (min/max) for this parameter.

Lock Threshold  
Indication and edit text  
2\textsuperscript{nd} fly wheel parameter; see row below

Ok  
Button  
Save the parameter of this tab to NT registry and close the window

Apply  
Button  
Save the parameter of this tab to the NT registry.

Cancel  
Button  
Undo parameter modification of this tab.

**Behaviour**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editable text</td>
<td>If users define a value outside its range, the user is informed about his/her mistake and the box contents is set to the previous valid value.</td>
</tr>
</tbody>
</table>

**Requirements**: US.1270US.1280
7.6.1.6 File Assembler

Description: This tab is part of the User station configuration control window and defines parameter for the File Assembler.

Additional comments:

## Window elements of File Assembler tab

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Option boxes</td>
<td>Define the mode of the File Assembler</td>
</tr>
<tr>
<td>Measurement Interval</td>
<td>Text and indication box</td>
<td>Defines the measurement period for one entry of the Data Monitor display (see Data Stream Monitor).</td>
</tr>
</tbody>
</table>
## Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Mode = Operational (assembling of L/HRIT files), VCDU Trace (fill trace buffer with received VCDU) Source Packet Trace (fill trace buffer with Assembled source packets)</td>
</tr>
</tbody>
</table>

## Requirements

- US.570
- US.660
- US.1290
### 7.6.1.7 Log

Description: This tab is part of the User station configuration control window and configure, which types of log messages are stored to the different log files.

Additional comments:

<table>
<thead>
<tr>
<th>WINDOW ELEMENTS OF LOG TAB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NAME</strong></td>
</tr>
<tr>
<td>Current log filter</td>
</tr>
<tr>
<td>(upper part of tab)</td>
</tr>
<tr>
<td>Log section</td>
</tr>
<tr>
<td>Filter selection</td>
</tr>
</tbody>
</table>
## Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modification of log filter</td>
<td>If the modification of a log filter is confirmed by the user, the updated filter is written to the ConfigDB (NT registry). The concerned components taking the new filter.</td>
</tr>
</tbody>
</table>

## Requirements:
7.6.2 Station Key Unit Access

Description: Allows users to issue commands to the SKU.
Additional comments: It is only a TEST tool

Window elements of Station Key unit access

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Number</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>Seed</td>
<td>0000000000000000000000000000</td>
<td></td>
</tr>
<tr>
<td>General Screen requirement</td>
<td>Screen</td>
<td>General Screen requirement</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Toolbar</td>
<td>Toolbars</td>
<td>Move, minimize, close. The window can't be resized manually.</td>
</tr>
<tr>
<td>Command</td>
<td>Combo box</td>
<td>List of available commands defined for the SKU</td>
</tr>
<tr>
<td>Parameter</td>
<td>List box</td>
<td>Allows the input of command parameter. The parameter type depends on the selected command.</td>
</tr>
<tr>
<td>Send</td>
<td>Button</td>
<td>The command and its parameter are passed to the SKU via the MUBM.</td>
</tr>
<tr>
<td>SKU Response</td>
<td>Text box</td>
<td>Contains the command response returned by the SKU or a message if something else goes wrong during command execution.</td>
</tr>
<tr>
<td>File-exit</td>
<td>Menu</td>
<td>Close this window</td>
</tr>
<tr>
<td>Help</td>
<td>Menu</td>
<td>Start help menu</td>
</tr>
</tbody>
</table>

**Behaviour**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send button</td>
<td>The command and its parameter are send to sku. Depending on the command, the answer of the sku takes some time. The answer is display in the Returned Result box in a readable way.</td>
</tr>
<tr>
<td>File-exit</td>
<td>The exit is executed without waiting for sku command completion.</td>
</tr>
</tbody>
</table>

**Requirements:** US.0870

US.0880
7.6.3  Edit Transfer Filter

Picture
<table>
<thead>
<tr>
<th>File Name</th>
<th>Edit Transfer, File Size (Enhies)</th>
<th>File Name</th>
<th>Edit Transfer, File Size (Enhies)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCP</td>
<td>Yes</td>
<td>MSG</td>
<td>Yes</td>
</tr>
<tr>
<td>DCP</td>
<td>No</td>
<td>DCP</td>
<td>No</td>
</tr>
<tr>
<td>MSG</td>
<td>No</td>
<td>DCP</td>
<td>No</td>
</tr>
<tr>
<td>DCP</td>
<td>No</td>
<td>DCP</td>
<td>Yes</td>
</tr>
<tr>
<td>WMO</td>
<td>No</td>
<td>EKI</td>
<td>No</td>
</tr>
<tr>
<td>EKI</td>
<td>No</td>
<td>DCP</td>
<td>No</td>
</tr>
<tr>
<td>WMO</td>
<td>No</td>
<td>EKI</td>
<td>No</td>
</tr>
<tr>
<td>EKI</td>
<td>No</td>
<td>DCP</td>
<td>No</td>
</tr>
<tr>
<td>MPE</td>
<td>No</td>
<td>MPE</td>
<td>No</td>
</tr>
<tr>
<td>MPE</td>
<td>No</td>
<td>MPE</td>
<td>No</td>
</tr>
<tr>
<td>WMO</td>
<td>No</td>
<td>WMO</td>
<td>No</td>
</tr>
<tr>
<td>WMO</td>
<td>No</td>
<td>WMO</td>
<td>No</td>
</tr>
</tbody>
</table>

**Legend:**
- **Yes**: File is included.
- **No**: File is excluded.

This table represents the selection criteria for various file names, indicating whether they are included or excluded based on certain conditions.
Description: The Edit Transfer Filter provides an editor for the generation and modification of the transfer filter. For one filter file different filter entries may be defined and stored. The created or modified transfer filter file must be activated in the Auto Transfer-tab of the Configuration Control display.

Additional comments:

### Window elements of Station Key unit access

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>General requirement</td>
<td>Screen General Screen requirement</td>
</tr>
<tr>
<td>Toolbar</td>
<td>Toolbars</td>
<td>Move, minimize, close. The window can't be resized manually.</td>
</tr>
<tr>
<td>FilterName</td>
<td>Indication text</td>
<td>Shows the open filter file name</td>
</tr>
<tr>
<td>Filter Size</td>
<td>Indication text</td>
<td>Shows the current no of entries in the display list</td>
</tr>
<tr>
<td>New File</td>
<td>Button (1)</td>
<td>Close an open filter and creates an empty list</td>
</tr>
<tr>
<td>Open File</td>
<td>Button (2)</td>
<td>Open an existing filter file</td>
</tr>
<tr>
<td>Save File</td>
<td>Button (3)</td>
<td>Store the contents of the list to a filter file</td>
</tr>
<tr>
<td>Delete File</td>
<td>Button (4)</td>
<td>Delete an existing filter file</td>
</tr>
<tr>
<td>New Entry</td>
<td>Button (5)</td>
<td>Insert a new filter entry to the list</td>
</tr>
<tr>
<td>Copy Entry</td>
<td>Button (6)</td>
<td>Copy the selected entry and insert in the list</td>
</tr>
<tr>
<td>Modify Entry</td>
<td>Button (7)</td>
<td>Modify the selected filter entry</td>
</tr>
<tr>
<td>Delete Entry</td>
<td>Button (8)</td>
<td>Remove the selected entry from the list</td>
</tr>
<tr>
<td>Print</td>
<td>Button(9)</td>
<td>Print the contents of the Transfer Filter</td>
</tr>
<tr>
<td>Help</td>
<td>Button (10)</td>
<td>Help</td>
</tr>
<tr>
<td>File-Print</td>
<td>Menu</td>
<td>Print the filter list to a system printer</td>
</tr>
<tr>
<td>File-close</td>
<td>Menu</td>
<td>Clear the filter list without saving the data</td>
</tr>
<tr>
<td>File-save as</td>
<td>Menu</td>
<td>Same as File-save, but the user is asked for a file name.</td>
</tr>
<tr>
<td>File-save</td>
<td>Menu</td>
<td>Save the filter list to the open filter file</td>
</tr>
<tr>
<td>File-open</td>
<td>Menu</td>
<td>Open an existing filter file</td>
</tr>
<tr>
<td>File-delete</td>
<td>Menu</td>
<td>Delete an existing filter file</td>
</tr>
<tr>
<td>File-exit</td>
<td>Menu</td>
<td>Close this window</td>
</tr>
<tr>
<td>Help</td>
<td>Menu</td>
<td>Start help menu</td>
</tr>
<tr>
<td>Filter pane</td>
<td>Table</td>
<td>This shows the filter entries sorted by the product mask.</td>
</tr>
</tbody>
</table>
### Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Entry</td>
<td>Position the mouse at an entry and click the mouse button once.</td>
</tr>
<tr>
<td>Modify entry</td>
<td>Additionally to the ModifyEntry is called if an entry is selected by a double mouse click.</td>
</tr>
<tr>
<td>Modify and new entry</td>
<td>Both opens a window called “Edit Filter Entry”. In case of the modify entry, the contents of the selected entry is copied. In case of the new entry, the editable fields are empty.</td>
</tr>
</tbody>
</table>

**Requirements:** US.1130US.1135
7.6.3.1 Edit Filter Entry

Description: Edit the fields of a filter entry.
Additional comments:.

Window elements of File Assembler tab

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product ID 1</td>
<td>Text box</td>
<td>Define a mask for the product id 1 of the annotation header</td>
</tr>
<tr>
<td>Product Id 2</td>
<td>Text box</td>
<td>Define a mask for the product id 2 of the annotation header</td>
</tr>
<tr>
<td>Product Id 3</td>
<td>Text box</td>
<td>Define a mask for the product id 3 of the annotation header</td>
</tr>
<tr>
<td>Processing Statge</td>
<td>Check boxes</td>
<td>Define which file matching the product id shall be transferred.</td>
</tr>
<tr>
<td>Transfer Destination</td>
<td>Text box</td>
<td>Set the destination directory on the remote computer.</td>
</tr>
<tr>
<td>Accept</td>
<td>Button</td>
<td>Insert/modify the contents of the fields to filter list, close the window and return to Edit Transfer Filter</td>
</tr>
<tr>
<td>Cancel</td>
<td>Button</td>
<td>Close the window and return to Edit Transfer Filter</td>
</tr>
</tbody>
</table>
Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>The content of the editable fields is checked. On error the user is requested to correct the error. If an entry is modified, its content is replaced. If a new entry is created, it is inserted to the filter list.</td>
</tr>
</tbody>
</table>

Requirements::
7.7 Log

The logging procedures are provided by the VCS log facility. For detailed description see [Annex B].

7.7.1 Realtime logview

Picture
Description: The monitoring of log messages generated by the user station software.

Additional comments:

Window elements of Realtime Logview

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Screen</td>
<td>Screen</td>
<td>General Screen requirement</td>
</tr>
<tr>
<td>Toolbar</td>
<td>Toolbars</td>
<td>Move, minimize, close. The window can’t be resized manually.</td>
</tr>
<tr>
<td>Keep window on top</td>
<td>Check box</td>
<td>If a tick is set in the box, the window is kept on the top of the screen</td>
</tr>
<tr>
<td>Clear Window</td>
<td>Button</td>
<td>Clear the message table</td>
</tr>
<tr>
<td>Stop/Continue</td>
<td>Button</td>
<td>Stop/continue the Log Server</td>
</tr>
<tr>
<td>DUMP</td>
<td>Button</td>
<td>This button is only usable, if the Log Server is stopped. The it dumps the contents of the log message display to a file. The name is hardcoded.</td>
</tr>
<tr>
<td>Quit</td>
<td>Button</td>
<td>Close the window</td>
</tr>
<tr>
<td>Log message list</td>
<td>Table</td>
<td>Display the incoming log messages</td>
</tr>
</tbody>
</table>

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop/Continue/DUMP</td>
<td>If the Log Server is running, this button is used to stop the log server. If the Log server is in the stop-state, this button is used to continue the log server. Only in this state the DUMP button is enabled.</td>
</tr>
</tbody>
</table>

Requirements: US.1220
7.7.2 Historical logview

Picture
Annex A: User Station MMIs
Description: Display the contents of a log file generated by the VCS Log facility.

Additional comments:

Window elements of View log files

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>General Screen requirement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>General Screen requirement</td>
</tr>
<tr>
<td>Toolbar</td>
<td>Toolbars</td>
<td>Move, minimize, close. The window can’t be resized manually.</td>
</tr>
<tr>
<td>Open</td>
<td>Button</td>
<td>Open an existing log file</td>
</tr>
<tr>
<td>Edit buttons</td>
<td>Button</td>
<td>Edit functions cut, copy and paste</td>
</tr>
<tr>
<td>Print</td>
<td>Button</td>
<td>Select a printer and print the contents of the open log file</td>
</tr>
<tr>
<td>About</td>
<td>Button</td>
<td>Shows information about the VCSlog</td>
</tr>
<tr>
<td>Update</td>
<td>Button</td>
<td>Update the display of the log file</td>
</tr>
<tr>
<td>Format</td>
<td>Button</td>
<td>Format the open log file</td>
</tr>
<tr>
<td>Delete</td>
<td>Button</td>
<td>Delete a log file</td>
</tr>
<tr>
<td>File</td>
<td>Menu</td>
<td>Open, close and delete log files; print and print preview for log files and print setup; list of previous open log files; exit</td>
</tr>
</tbody>
</table>
| Edit | Menu | Undo, cut, copy and select all
Find, find next and replace
Update and format log files |
| View | Menu | Configuration of window |

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Requirements: US.1220
7.8 File Manager

The File Manager is an instance of the NT explorer. It is extended by assignments between L/HRIT files to generic viewer, manual decryption and manual decompression and assignments between subsampled file to image view.

Behaviour

<table>
<thead>
<tr>
<th>NAME</th>
<th>ITEM BEHAVIOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>View L/HRIT files</td>
<td>Select the XRIT file by a double-click</td>
</tr>
<tr>
<td>Historical Image view</td>
<td>Select the subsampled image file by a double-click</td>
</tr>
<tr>
<td>Trace Packet viewer</td>
<td>Select the trace buffer file by a double-click</td>
</tr>
<tr>
<td>Manual decryption</td>
<td>Select an XRIT File, open the context box with the right mouse button and select decryption.</td>
</tr>
<tr>
<td>Manual deompression</td>
<td>Select an XRIT File, open the context box with the right mouse button and select decompression.</td>
</tr>
<tr>
<td>Manual transfer to DADF Offline</td>
<td>Map the DADF offline system to a local device and use the window system file commands.</td>
</tr>
</tbody>
</table>

Requirements:: US.0100 US.0740 US.0750 US.0890
US.0970 US.1210
8 Annex B: VCS Log

8.1 Introduction

This document describes the VCS logging mechanism implemented with the operating system Windows NT. It consists of the three following sections:

- Implementation
- Interface Specification
- Tools

For installing VCSLOG please refer to the VCSLOG/VCSSUPER Installation Guide.

8.2 Implementation

This section is divided into the three following parts:

- Introduction
- Programming the DLL
- Programming the Server

8.2.1 Introduction

In the implementation essential functions of the operating system Windows NT are used:

- Process Synchronization
- Memory Mapped Files
- Dynamic Link Libraries

8.2.1.1 Process Synchronization

The program library of Windows NT offers different possibilities to synchronize parallel active processes or threads. These are the most important of them:

- Events
- Critical Sections
- Mutexes

In the implementation of the VCS logging mechanism mutexes are used that allow synchronization on process level as well as on thread level.

8.2.1.2 Memory Mapped Files

Windows NT allows to map files on sections in the virtual memory (there is 2GByte free available memory, protected against other processes). You do not access to the contents of these mapped files by the usual file operations. You rather use a pointer to the memory section which is handled like a normal memory access.

Memory mapped files can be used by several processes at the same time. The operating system has the task to synchronize the contents of the mapped files. The synchronization of read and write accesses can be accomplished by the above-mentioned mutexes.

In the VCS logging mechanism simultaneous access by several processes to a memory mapped file is accomplished by means of the following actions:

- The messages of a process are stored efficiently without performance intensive file operations, because the writing process believes to write the messages in its virtual memory.
- Any number of processes can use a central file simultaneously which sequences all messages by an appropriate operation.
• A central process reads the simultaneously used files and stores the messages in process specific message files. The memory mapped file is used as a ring buffer.

8.2.1.3 Dynamic Link Libraries

The operating system Windows NT offers the possibility to administrate segments of programs with libraries. Not until the function of the program segment is needed it will be linked to the memory section of the process. The resource requirements of the processes are minimized because

• when starting the program, first, only those program segments will be loaded which are needed.
• program segments swapped out in DLLs can be used by several processes at the same time (code sharing). The operating system maps memory sections used locally by the DLL (e.g. static variables) to the local memory section of each process by means of a COPY-ON-WRITE mechanism.

In the implementation of the logging mechanism a DLL is used which allows simultaneous write access to one memory mapped file by means of an exporting function.

8.2.2 Programming the DLL

The programming of the DLL is described in the following sections:

• Initialization
• Administrative Information
• Implementing the Access to a Memory Mapped File

8.2.2.1 Initialization

Each DLL in Windows NT contains the special function DllMain, which will be invoked automatically by the operating system in the following situations:

• first invocation of a routine of the DLL of a process
• first invocation of a routine of the DLL of a thread
• release of the DLL by a thread (automatically when terminating the thread, manually by invocation of e.g. FreeLibrary)
• release of the DLL by a process (automatically when terminating the process, manually by invocation of e.g. FreeLibrary)

The routine DllMain is used to initialize and de-initialize a simultaneous access to the memory mapped file.
During the Initialization the following steps are performed:

- **First call of a DLL-routine**
- **VCSLOG is registered in the registry database**
- **Open file that is registered in the registry database**
- **Inform operating system that the file shall be mapped in the memory**
- **Set NOINIT flag**
- **Set INIT flag**

Node 1 causes the following side effect: If a memory mapped file does not yet exist it will be created. Size and position of the file depends on the configurations in the registry.
During the de-initialization the following steps are performed:

- **INIT FLAG SET**
  - Inform operating system that the file shall not be mapped in the memory any longer.
  - Close file

### 8.2.2.2 Administrative Information

The memory mapped file has the following structure:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextFreeLine</td>
<td>Integer, 4 Byte</td>
<td>1</td>
<td>Index of the next free row</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Client - DLL's: write access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Server: read access</td>
</tr>
<tr>
<td>LastLineWritten</td>
<td>Integer, 4 Byte</td>
<td>1</td>
<td>Index of the last written row</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Client - DLL's: read access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Server: write access</td>
</tr>
<tr>
<td>LogLines</td>
<td>Structure: filename, Char, 260 Bytes moduleName, Char, 6 Bytes level, Char, 1 Byte mnemonic, Char, 6 Bytes text, Char, 81 Bytes time, structure, 16 Byte code, Integer, 4 Byte threadId, ulong, 4 Byte processName, char, 61 Bytes</td>
<td>500 - 10000 configurable</td>
<td>Each written message has the stated structure</td>
</tr>
</tbody>
</table>

**Mutexes:**

Name: "NTLOG_LOG_MESSAGE_BUFFER_MUTEX"

It is used to synchronize the access to logBuffer. Immediately after access the mutexes are released. Presently, this mutex is used by the following applications:

- VSLOGSV.EXE
- RLM_UI.EXE
- WLM.DLL
8.2.2.3 Implementing the Access to a Memory Mapped File

The write access to a memory mapped file is realized by the following steps:

- **INIT flag not set**
  - Get actual system time for logtimestamp
  - Open handle on Mutex
  - Generate logrow
  - Reserve mutex
  - Reservation finished
  - Get next free logrow in logbuffer
  - Write row in logbuffer
  - Release mutex

- **Timeout 500msec**
  - Generate logrow
  - Open handle on Mutex

8.2.3 Implementing the Server

The server has the task to write the messages, stored in the memory mapped file, into the logfiles stated in the corresponding message.
The log messages are written into a directory specified in the registry entry
HKEY_LOCAL_MACHINE \ Software \ VCS \ VCSLOG \ LOCAL_LOG_DIRECTORY

8.2.3.2 Writing Log Messages to Network Directories

To control processes on a network by means of log messages, all appropriate messages must be written to a common network device. All messages written to it can be analyzed by a central tool. This tool can perform appropriate actions.

8.2.3.2.1 Implementation

The logserver analyzes the loglevel of the incoming message. If this loglevel belongs to the set of loglevels, which shall be written into the network directory, the incoming message will be written to the network directory in addition to the local directory.

If no network directory was specified, this analysis is dropped.

8.2.3.2.2 Configuration

The logserver writes all log messages with a loglevel belonging to a set of loglevels defined in the registry entry HKEY_LOCAL_MACHINE \ Software \ VCS \ VCSLOG \ NET_LOG_TYPES into a directory specified in the registry entry HKEY_LOCAL_MACHINE \ Software \ VCS \ VCSLOG \ NET_LOG_DIRECTORY
8.3 Interface Description

The interface of the log mechanism consists of the following routines which have to be integrated with the user processes.

8.3.1 writeLogMessage

Probably the most important routine of this DLL:

```c
int writeLogMessage ( const char *fileName,
                     const char *module,
                     const char status,
                     unsigned long code,
                     const char *mnemonic,
                     const char *text);
```

with

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>Name of the logfile the message shall be written into. Maximal 260 characters are allowed. Specification of a drive is ignored. Suffixes (e.g. .TXT) are ignored. The suffix .LOG is added automatically. The specified file is always stored in the specified directory. Log files are always stored in the same directory where the logbuffer file is stored. Any subdirectories are created always herein.</td>
</tr>
<tr>
<td>module</td>
<td>Name of the module that generates the log message. Maximal 6 characters are allowed.</td>
</tr>
<tr>
<td>status</td>
<td>Status of the message: The following characters are supported: 'A' − 'Z', '0' − '5'</td>
</tr>
<tr>
<td>code</td>
<td>Integer error code</td>
</tr>
<tr>
<td>mnemonic</td>
<td>Mnemonic code of the error message. Maximal 6 characters are allowed.</td>
</tr>
<tr>
<td>text</td>
<td>Message text. Maximal 80 characters are allowed.</td>
</tr>
</tbody>
</table>

The routine uses the following synchronization objects:

Mutexts:

Name: "NTLOG_LOG_MESSAGE_BUFFER_MUTEX"

It is used to synchronize accesses to the log buffer. Immediately after access the mutex is released. At present this mutex is used by the following applications:

- VSLOGSV.EXE
- RLM_UI.EXE
- WLM.DLL

Events:

Name: "VCS_LOG_MESSAGE_EVENT_WRITE_MESSAGE"

After each message written successfully this AUTO-RESET event is set. This event is used by the following applications:
8.3.2 writeLogTimeMessage

A second routine is offered. It expects additionally a parameter of type `SYSTEMTIME *`.

```c
DLL(int) writeLogTimeMessage ( char *fileName,
    char *module,
    char status,
    unsigned long code,
    char *mnemonic,
    char *text,
    SYSTEMTIME *lpSystemTime);
```

This routine does not write the actual system time into the log buffer when writing the message, but a time value specified in `lpSystemTime`.

All other is the same as `writeLogMessage`.

8.3.3 GetLogLevel

The routine `DLL(unsigned long) GetLogLevel(LPCSTR lpcLogLevel)` converts the string specified in `lpcLogLevel` into bit notation. Each bit set corresponds to a loglevel which is found in the specified string.

By means of this bit notation it is possible to check, before creating a log message, whether the message is within the expected loglevel (with the friendly support of the routine `IsLog` described in chapter 8.3.4).

Example:

```c
unsigned long g_ulLogLevel;

void initLogLevel ()
{
    char *lpcLogLevel;

    lpcLogLevel = "AEFWIHS"
    g_ulLogLevel = GetLogLevel (lpcLogLevel);
}

... int wlm (..., char cLogLevel, ....)
{
    if (IsLog (g_ulLogLevel, cLogLevel))
    {
        writeLogMessage (...);
    }
}
...
8.3.4 IsLog

The routine `DLL(BOOL) IsLog(const unsigned long lLevel, const char status)` checks whether the loglevel specified in `status` exists in the bit notation `lLevel`.
8.4 Tools

For the operational usage of the logging mechanism the following tools are necessary; they are delivered together with the logserver and the DLL:

- CLEANUP
- LOGVIEWER
- VCS_SUPER
- RLM_UI

8.4.1 CLEANUP

The process CLEANUP examines all logfiles which it finds in the log directory and its subdirectories. Corresponding to the loglevels of each message in the log files old messages are deleted from the log files.

The residence time of log messages of a specific loglevel is defined by the registry entry:

```
HKEY_LOCAL_MACHINE\Software\VCS\VCSLOG\CLEANUP\KEEP_MESSAGES
```

The residence time is stated in the following way:

<table>
<thead>
<tr>
<th>ValueName</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;n&gt;_any text</td>
<td>&lt;loglevel&gt;[&lt;loglevel&gt;]</td>
</tr>
</tbody>
</table>

WITH

- **n** integer, number of days of residence time. The character '_' behind the integer is obligatory.
- **loglevel** 'A'-'Z' '0'-'9'

The notation `<loglevel>[<loglevel>]` denotes „one or more loglevels“.

Examples:

```
HKEY_LOCAL_MACHINE\Software\VCS\VCSLOG\CLEANUP\KEEP_MESSAGES

1_DAY IHXNO
2_DAYS A
3_DAYS W
4_DAYS E
5_DAYS FS
```

Log messages with loglevel I (Interactive), H (Heartbeat), X (program specific extension), N (NORMAL), O (Other) reside max. one day in the logfile.

Log Messages with loglevel A (Automatic) reside max. two days in the logfile.

Log Messages with loglevel W (WARNING) reside max. three days in the logfile.

Log Messages with loglevel E (ERROR) reside max. five days in the logfile.

Log Messages with loglevel F (FATAL) and S (Start/Stop) reside max. five days in the logfile.

All other messages reside max. one day in the logfile.

The process is called once a day by the logserver process, which waits 5 minutes (as default, configurable) for the termination of CLEANUP.
8.4.2 LOGVIEWER

In the contrary to the above-mentioned programs LOGVIEWER is an interactive process. It allows to display the loglevel specific formatting and to print logfiles.
8.4.3 VCS_SUPER

As it is necessary that log messages can be written anytime into the corresponding files, the process VCSLOGSV (of the logserver) must run permanently. The process VCS_SUPER controls this process and starts it if it cannot be found on the system.

Please refer to the VCSSUPER Referenzhandbuch for details.

8.4.4 RLM_UI

This tool displays the incoming messages in realtime. The messages are displayed in a chronological sequence in a list. The size of the list is configurable.

- Delete all existing messages
- Write all messages into a file
- Stop displaying
The following message components can be blinded out:

- Mnemonic
- Module
- Date
- Thread ID
- Process name

You may define a filter to display only messages with a specified loglevel. Additionally if desired, a keyboard beep signals all incoming messages of a specified loglevel.

If active, only messages of a specified file are displayed.
9 Appendix C: VCSSUPER Reference Manual

9.1 Introduction

Windows NT supports preemptive multitasking both on process level and thread level. Each process running within its own virtual address range can be started independently of each other so that they may run parallel.

Windows NT supports automatic start of processes after booting the operating system independently of users logged in (so-called services). But Windows NT does not offer a supervising function to detect whether a process has closed, and to restart it automatically.

The process VCSSUPER specified in this manual has the task to start and supervise Windows NT processes. After closing of a process that was started by VCSSUPER, it will be restarted by VCSSUPER.

9.2 Implementation

In the following sections the implementation of VCSSUPER is described.

9.2.1 Making Use of the Operating System

The application programmer may use all functions the operating system offers by means of the libraries which are collected in WIN32. The following services are of special interest for VCSSUPER:

- Programming of processes
  Especially starting, closing, controlling and performance management
- Setting up services
- Registry database

9.2.1.1 Processes

9.2.1.1.1 Introduction

In Windows NT a process is a program that runs within its own virtual address range. Each process is identified by a unique number, the process handle. Additionally each running process is given a name (not unique), which results from the name of the program that was started.\(^1\)

Example:
Opening the notepad from the group Accessories the program manager starts the program

\[D:\WINNT35\SYSTEM32\NOTEPAD.EXE\] (The directory is only an example. It may differ depending on the installation.)

The operating system assigns a unique number (e.g. 0x4c) to the process started. The name of the process is NOTEPAD, i.e. the name of the program started, without directory and extension. If the notepad will be opened once again, a new number (e.g. 0x5d) will be given to the new process, but the name of the process will be again NOTEPAD.

9.2.1.1.2 Starting Processes

By means of the library routine CreateProcess application programs can start new processes. For this purpose among others the following parameters must be set:

- name of the application that is to start
- command row
- start flags

\(^1\) See MSVC++ 4.x Online Documentation: SDK's\Win32 SDK\Win32 Programmers Reference\Overviews\SystemServices\Processes and Threads
• working directory
• startup information

The identification number of the process started is reported to the calling process. This number is kept unique in the system until the calling process resets it by calling the routine `CloseHandle`.

This important property is used implementing VCSSUPER.2

9.2.1.3 Controlling

The library routine `CreateProcess` supplies information about the actual status of a process. If the process, identified by its number, is still active, the value `STILL_ACTIVE` will be delivered. If the process has already terminated the return status is delivered.3

9.2.1.4 Performance Management

WIN32 does not offer routines which supply information about the actually running processes. But you can get this information indirectly by calling the data accumulated for the performance management4. By means of a special entry HKEY_PERFORMANCE_DATA you can create from the registry database the list of the actually running processes. Details about the implementation of this data determination are not given here, because it was taken from the `ProcessViewer`. For more information please refer to the sources to this program which are delivered by Microsoft in the Microsoft Developer Networks (section SDK`s)5.

Addition to Windows NT 4.0:

Since Windows NT 4.0 the PSAPI.DLL is part of the Windows NT Software Development Kits. This DLL offers a more comfortable access to the data administrated by the performance management. Central routines are

• `EnumProcesses`
  Supplies the Id of each running process. This Id can be used for the WIN32 call `OpenProcess`

• `EnumProcessModules`
  Supplies the Id of each DLL, that participate in one process

• `GetModuleBaseName`
  Supplies the name of a specified module

By these routines you can easily create queries whether a special process is active or not.

9.2.1.2 Services

Services are programs which start automatically after boot of the operating system. These services are available independent of users logged in. For instance SCSI driver or network services shall be mentioned here.

By means of the entry `Services` in the ControlPanel you can suspend, start or completely disable services. The operating system does not restart closed services.

The Windows NT 4.0 Kit contains a program usable as a batch program named SC.EXE, which facilitates starting, closing and controlling services.

---

2 See MSVC++ 4.x Online Documentation: SDK's\Win32 SDK\Win32 Programmers Reference\Overviews\SystemServices\Processes and Threads
3 See MSVC++ 4.x Online Documentation: SDK's\Win32 SDK\Win32 Programmers Reference\Overviews\SystemServices\Processes and Threads
4 See MSVC++ 4.x Online Documentation: SDK's\Win32 SDK\Win32 Programmers Reference\Overviews\SystemServices\Processes and Threads
5 After Installation of the Samples of the SDK in the following directory:
\*:\MSTOOLS\samples\win32\winnt\pviewer
The program VCSSUPER is installed as a service, so it is available to all users. Additionally it is possible to switch it comfortably on or off.

**9.2.2 Controlling VCSSUPER**

The program VCSSUPER is controlled by entries in the registry database.

**9.2.2.1 Registry Database**

The registry database of Windows NT offers among other major keys the entry \texttt{HKEY\_LOCAL\_MACHINE}. Here all data shall be entered which refer to the work station.

Within the key \texttt{HKEY\_LOCAL\_MACHINE\\SOFTWARE\VCS\VCSSUPER} all information about controlling VCSSUPER shall be entered.\(^6\)

**9.2.2.1.1 Logging on VCSSUPER**

For each process that is to supervise you have to create its own key and set therein the corresponding parameters as values.

The keys must be set as follows:

\[
\begin{align*}
\text{HKEY\_LOCAL\_MACHINE\\SOFTWARE\VCS\VCSSUPER\<program name1>}
\text{HKEY\_LOCAL\_MACHINE\\SOFTWARE\VCS\VCSSUPER\<program name2>}
\end{align*}
\]

...  

The following parameters are necessary:

- location
- command line
- priorities (mainprio, subprio)
- supervising flag
- working directory

The key hierarchy looks as follows:

```
/-----------key-------------------------------/ --value--\ 
HKEY\_LOCAL\_MACHINE\\SOFTWARE\VCS\VCSSUPER\<program name1>\Location
HKEY\_LOCAL\_MACHINE\\SOFTWARE\VCS\VCSSUPER\<program name1>\Commandline
HKEY\_LOCAL\_MACHINE\\SOFTWARE\VCS\VCSSUPER\<program name1>\Prio
HKEY\_LOCAL\_MACHINE\\SOFTWARE\VCS\VCSSUPER\<program name1>\WorkingDirectory
HKEY\_LOCAL\_MACHINE\\SOFTWARE\VCS\VCSSUPER\<program name1>\Supervised
```

**Location**

The location specifies the access directory to the program that shall be supervised. It can be set as follows:

<table>
<thead>
<tr>
<th>Name of the value</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>&lt;drive&gt;...\PROGRAM NAME</td>
<td>This kind of notation shall be used in case of programs which have not been written by VCS.</td>
</tr>
</tbody>
</table>
| Location          | HKEY\[LOCAL\_MACHINE]\<key directory in the registry database> or HKEY\[LOCAL\_MACHINE]\<key directory in the registry database> | This kind of notation shall be used in case of programs which have been written by VCS, because for these programs you have to set their access directory in the registry database. When shifting a program you only have to change the original.

---

\(^6\) See MSVC++ 4.x Online Documentation: SDK\\s\Win32 SDK\Win32 Programmer's Reference\Overviews\SystemServices\Registry
<table>
<thead>
<tr>
<th>Name of the value</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKEY\CURRENT_USER&lt;key directory in the registry database&gt; or HKEY\USERS&lt;key directory in the registry database&gt;</td>
<td>location.</td>
<td></td>
</tr>
</tbody>
</table>

If this parameter cannot be found in the registry database the process will abort.

Examples:

Key:
```
HKEY_LOCAL_MACHINE\SOFTWARE\VCS\VCSSUPER\VCSLOG\SERVER
```
Value:
```
Location: D:\vcslog\logserver.exe
```
*The specified program is called directly.*

Key:
```
HKEY_LOCAL_MACHINE\SOFTWARE\VCS\VCSSUPER\VCSLOG\SERVER
```
Value:
```
Location: KEY\LOCAL_MACHINE\SOFTWARE\VCS\VCSSUPER\VCSLOG\CLEANUP\Location
```
*The value of the specified registry entry is read, then the therein stated program starts.*

**Command Line**
The specified character string is transferred as a command line to the starting program. The syntax is as follows:

<table>
<thead>
<tr>
<th>Name of the value</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commandline</td>
<td>DEFAULT</td>
<td>An empty command line is transferred.</td>
</tr>
<tr>
<td>Commandline</td>
<td>&lt;any character string&gt;</td>
<td>The specified character string is transferred as a command line.</td>
</tr>
</tbody>
</table>

If this parameter cannot be found in the registry database the value DEFAULT will be assumed as default.

**Priority**
There are four alternatives to specify the priority of the process that is to start.

<table>
<thead>
<tr>
<th>Name of the value</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prio</td>
<td>IDLE</td>
<td>Lowest priority. A process with this priority is only running if the system is idle. This is the ideal priority for background tasks like screen saver or cleaner.</td>
</tr>
<tr>
<td>Prio</td>
<td>NORMAL</td>
<td>Normal priority.</td>
</tr>
<tr>
<td>Prio</td>
<td>HIGH</td>
<td>High priority should only be used for actually important activities. Busy loops might be possible and are difficult to cancel as the Task Manager runs with the same priority and might have no chance.</td>
</tr>
<tr>
<td>Prio</td>
<td>REALTIME</td>
<td>Highest priority takes precedence over all activities of the operating system like caching, mouse</td>
</tr>
</tbody>
</table>
9.2.2.1.2 Logging off

There are two ways to inactivate the supervising of a process.

- Set the Parameter SUPERVISED to FALSE
- Delete the complete key of the process

9.2.3 Implementing VCSSUPER

1.) Starting VCSSUPER the configuration parameters of VCSSUPER are read.

2.) A list of the keys is created which you can find in the following directory in the registry database:
HKEY_LOCAL_MACHINE\SOFTWARE\VCS\VCSSUPER

3.) Each key is executed as follows:

3a.) Is the parameter SUPERVISED on TRUE? Yes, go on with 3b) or else 4).

3b.) Is the directory specified correctly? Yes, go on with 3c) or else 4).

3c.) Determine the name of the process.

3d.) Is the process still active? Yes, go on with 3.) or else 3e)
- If the process was started by VCSSUPER it is searched for the name of the process in the internal list of supervised processes. The handle of the process saved in this list is used when calling GetExitCodeProcess.

- If name of the process cannot be found in the internal list, the process list of the operating system must be searched through.

3e.) Read the start parameters of the process.

3f.) Start the new process. If successfully started enter the name and the handle in the internal process list.

3g.) Go to 3.

4.) Wait N seconds as specified in the configuration parameter.

5.) Go to 2.