SKA1 Mid.CBF Master Control Software Design Report

A description of Minimum Viable Product at the end of Program Increment #7

TALON Document Number T1-DRE-00450
File Type: ..... DRE
Revision: 1C
Author ..... An Yu
Date 2020-08-20
Status..... Draft
Classification..... Unrestricted

Prepared By	Name	An Yu	Signature	
	Organisation	NRC Canada		
Reviewed By	Name	Sonja Vrcić	Signature	
	Organisation	NRC Canada		
	Name		Signature	
	Organisation			
Approved By	Name	Mike Pleasance	Signature	
	Organisation	NRC		
Issued By	Name	Donna Morgan	Signature	
	Organisation	NRC		

Document History

Revision	Date	Changes/Notes	Author
A	2019-08-14	Initial version.	J. Jiang
B	2019-08-20	Added document numbers and issue dates to applicable documents. Added requirements and assumptions. Added state diagrams for each software component. Made other minor changes.	J. Jiang
1	2019-09-03	Changed doc type to Design Report. Added doc #, updated file name and title page	D. Morgan
1A	2020-04-22	Updated document to include the new layout for fspSubarrays and fsp devices	R. Voigt
1B	2020-05-25	Updated and simplied steps in Development Environment and Starting Guide. Included more explaination. Corrected some erros in the installation process	A. Yu
1C	2020-08-21	Merged starting guide and Development environment chapter. Included a more comprehensive guide on running the software, different developer tools, and tips for developers. Modified all the parts that are related to the state machine, configurScan JSON file, outputlink description changes made in PI7. Updated command and attribute changes.	A. Yu

Table of Contents

List of Figures

List of Tables

Terms, Acronyms, and Abbreviations

CS	Control System
CSP	Central Signal Processor
CBF	Correlator and Beamformer
FSP	Frequency Slice Processor
FQDN	Fully Qualified Domain Name
GUI	Graphical User Interface
HPS	Hard Processor System
ICD	Interface Control Document
JSON	
LMC	Local Monitor and Control
LRU	Line Replaceable Unit
FS	Frequency Slice
FSP	Frequency Slice Processor
MCS	Master Control Software
Mid	SKA1 Mid-Frequency Telescope
MVP	Minimum Viable Product
PSS	Pulsar Search
PST	Pulsar Timing
SAFe	Scaled Agile Framework
SDP	Science Data Processor
SKA	Square Kilometre Array
SKA1	SKA Phase 1
SKAO	SKA Organization
TANGO	Control System Framework
TMC	Telescope Management and Control
VCC	Very Coarse Channeliser
VLBI	Very Long Baseline Interferometry

1 Introduction

This document describes the SKA1 Mid.CBF Master Control Software (MCS) developed as part of the Evolutionary Prototype for Program Increment #3, referred to as the Minimum Viable Product (MVP).

During the design phase, several interfaces departed from existing Interface Control Documents and Detailed Design Documents and were further refined, as detailed in CHAPTER 14.

This work was performed within the context of the SKAO Scaled Agile Framework (SAFe) Release Train, using Atlassian tools Confluence and JIRA for inter- and intra-team communication.

2 Applicable and Reference Documents

2.1 Applicable Documents

The following documents at their indicated revision form part of this document to the extent specified herein. Unless otherwise noted, the latest Revision is assumed.

Table 2-1 Applicable Documents

Ref No	Document/Drawing Number	Document Title	Issue Number
AD1	SKA-TEL-CSP-00000066	Mid.CBF DDD	11 October 2018
AD2	SKA-TEL-CSP-0000019	ICD CSP.LMC to Mid.CBF	26 June 2018
AD3	000-000000-010	SKA CS Guidelines	5 May 2018

2.2 Reference Documents

The following documents provide useful reference information associated with this document. These documents are to be used for information only. Changes to the date and/or revision number do not make this document out of date.

Table 2-2 Reference Documents

Ref No	Document/Drawing Number	Document Title	Issue Number
RD1	ECSS-E-ST-40C	European Cooperation for Space Standardization, Space Engineering - Software	6 March 2009

3 Overview and Context

3.1 Function and Purpose

The Mid.CBF MCS implements TANGO Devices that facilitate high-level control of Mid.CBF by the CSP Local Monitor and Control (CSP.LMC), and consequently by the Telescope Management and Control (TMC). It provides an interface that acts as an intermediate between CSP.LMC and the TALON-DX boards that implement signal processing functionality.

3.2 Environment considerations

No particular considerations were given to the operating environment. Refer to CHAPTER 10 for environment details.

3.3 Context

The Mid.CBF MCS was developed as part of the MVP (FIGURE 3-1). The primary goal of the MVP is to be able to create, with a text editor, a simple scheduling block that is executed by the Observation Execution Tool (OET), resulting in control over the following interfaces:

- TMC to DISH
- TMC to SDP
- TMC to CSP (including Mid.CBF MCS)

The scheduling block should execute a simple scan where:

- The telescopes are configured to slew to a source at a nominal RA, Dec
- The SDP is configured to accept data
- The correlator is configured to have a simple imaging mode

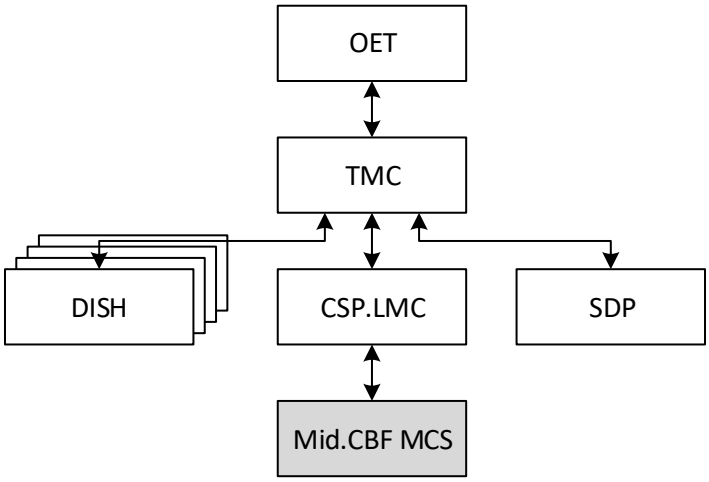

Figure 3-1 Breakdown of the MVP

The CSP.LMC exposes a set of attributes and commands, allowing TMC to change its operational state, add and release resources to a particular sub-array, configure a scan for imaging, and start, execute, and end a scan. These commands are forwarded to Mid.CBF MCS, where appropriate actions are taken, though nothing is actually executed on hardware.

For completeness, the context diagram of Mid.CBF MCS and TALON-DX boards is given in FIGURE 3-2. In the future, the Mid.CBF MCS will communicate with the TALON LRU Control and TALON-DX Master TANGO Devices.

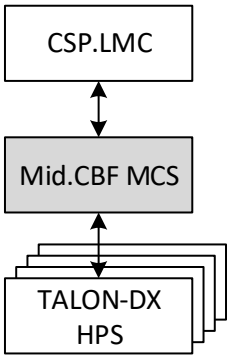

Figure 3-2 Context diagram of Mid.CBF MCS and TALON-DX boards

3.4 Constraints

No constraints to development were identified.

4 Requirement Specifications

4.1 Functional requirements

Table 4-1 Functional Requirements

ID	Requirement
Mid.CBF-MCS-PI3-REQ-001	Upon successful initialization Mid.CBF Master shall transition to operational state STANDBY.
Mid.CBF-MCS-PI3-REQ-002	Mid.CBF Master shall implement commands, attributes and functionality as described in Mid.CBF DDD [AD1] and ICD Mid.CBF to CSP.LMC AD2] required to bring Mid.CBF MCS in operational state ON.
Mid.CBF-MCS-PI3-REQ-003	Mid.CBF Master shall implement commands, attributes and functionality as described in Mid.CBF DDD [AD1] and ICD Mid.CBF to CSP.LMC [AD2] required to bring Mid.CBF MCS in operational state OFF.
Mid.CBF-MCS-PI3-REQ-004	Mid.CBF Master Control Software Release PI#3 shall instantiate a single subarray.
Mid.CBF-MCS-PI3-REQ-005	Mid.CBF Master Control Software Release PI#3 shall instantiate four VCCs.
Mid.CBF-MCS-PI3-REQ-006	Mid.CBF Master Control Software Release PI#3 shall randomly assign receptor IDs to VCCs.
Mid.CBF-MCS-PI3-REQ-007	Mid.CBF Master Control Software Release PI#3 shall instantiate four FSPs.
Mid.CBF-MCS-PI3-REQ-007	Mid.CBF Subarray shall implement command add receptors to the subarray.
Mid.CBF-MCS-PI3-REQ-009	Mid.CBF Subarray shall implement command remove receptors from the subarray.
Mid.CBF-MCS-PI3-REQ-010	Mid.CBF Subarray shall implement command configure scan for correlation as defined in ICD Mid.CBF to CSP.LMC [AD2].
Mid.CBF-MCS-PI3-REQ-011	Mid.CBF Subarray shall implement functionality required for transition to observing state SCANNING. The functionality and message content is defined in ICD Mid.CBF to CSP.LMC [AD2], amendments are defined on SKA Confluence: https://confluence.skatelescope.org/display/SE/TM+to+CSP+ICD
Mid.CBF-MCS-PI3-REQ-012	Mid.CBF Subarray shall implement command start scan.
Mid.CBF-MCS-PI3-REQ-013	Mid.CBF Subarray shall implement command end scan.
Mid.CBF-MCS-PI3-REQ-014	Mid.CBF Subarray shall implement observing state transitions as defined in Mid.CBF DDD [AD1].

ID	Requirement
Mid.CBF-MCS-PI3-REQ-015	Mid.CBF MCS TANGO Device VCC shall support at least the functionality required to support Mid.CBF transition to operational state ON, scan configuration for correlation, scan start and scan end.
Mid.CBF-MCS-PI3-REQ-016	Mid.CBF MCS TANGO Device FSP shall support at least the functionality required to support Mid.CBF transition to operational state ON, scan configuration for correlation, scan start and scan end.
Mid.CBF-MCS-PI3-REQ-017	Mid.CBF Subarray shall implement command go to idle.

4.2 Performance requirements

Specific performance requirements for this initial version of the Mid.CBF MCS have not been identified, therefore formal requirements related to performance of Release PI#3 have not been specified.
The following performance is acceptable:

a) Software initialization is completed in less than 15 seconds.

b) Each command is executed in less than 5 seconds.

Note that when TALON-DX hardware is deployed, execution of commands that require re-configuration of FPGAs may take longer to complete; this version however implements only a subset of high-level TANGO devices and state transitions are almost instantaneous.

4.3 Interface requirements

Table 4-2 Interface requirements

ID	Requirements
Mid.CBF-MCS-PI3-REQ-030	Mid.CBF Master Control Software shall implement interface with Telescope Monitor and Control as defined in the Interface Control Document CSP.LMC to Mid.CBF [AD2] and amendments defined on the SKA Confluence pages at this link: https://confluence.skatelescope.org/display/SE/Key+Interface+descriptions Note: In PI#3 Release all commands are implemented as synchronous commands.
Mid.CBF-MCS-PI3-REQ-031	Mid.CBF Master Control Software shall provide GUI interface that displays all major state and mode indicators for TANGO devices implemented as a part of PI#3 Release.

Mid.CBF software running on TALON-DX board is not ready to support communication with the Master Control Software, therefore, implementation of the interface with the TALON-DX Board software is not required.

4.4 Operational requirements

Table 4-3 Operational requirements

ID	Requirement
Mid.CBF-MCS-PI3-REQ-041	Mid.CBF MCS shall run within containerised environment as specified on the SKA Developer Portal. http://developer.skatelescope.org/en/latest/
Mid.CBF-MCS-PI3-REQ-042	Mid.CBF Master and Subarray shall implement operational state and other state and mode indicators defined in Mid.CBF Detailed Design Document [AD1].
Mid.CBF-MCS-PI3-REQ-043	Mid.CBF TANGO Devices for which detailed list of modes and states is not defined in Mid.CBF Detailed Design Document [AD1] shall implement mode and state indicators as defined for Capabilities in the document SKA Control System Guidelines [AD3].

4.5 Resources requirements

Specific requirements related to resources used by Mid.CBF MCS have not been identified, other then the requirement that PI#3 Release of Mid.CBF MCS software can be executed on a developer workstation and that CI tests can be executed on the server provided by the SKA organization.

4.6 Design requirements and implementation constraints

Table 4-4 Design requirements

ID	Requirement
Mid.CBF-MCS-PI3-REQ-061	Mid.CBF MCS shall run within containerised environment as specified on the SKA Developer Portal. http://developer.skatelescope.org/en/latest/
Mid.CBF-MCS-PI3-REQ-062	Mid.CBF Master and Subarray shall implement operational state and other state and mode indicators defined in Mid.CBF Detailed Design Document [**AD1**].
Mid.CBF-MCS-PI3-REQ-063	Mid.CBF TANGO Devices for which detailed list of modes and states is not defined in Mid.CBF Detailed Design Document [AD1] shall implement mode and state indicators as defined for Capabilities in the document SKA Control System Guidelines [AD3].

4.7 Security and privacy requirements

PI#3 Release of Mid.CBF Master Control Software is not required to address security and privacy concerns.

4.8 Portability requirements

Portability is achieved by Mid.CBF-MCS-PI3-REQ-061.

4.9 Software quality requirements

Mid.CBF MCS shall implement a set of tests to be executed each time the software is submitted in the software library (on push). The CI guidelines are provided on the SKA Developer Portal.

Table 4-5 Software quality requirements

ID	Requirements
Mid.CBF-MCS-PI3-REQ-090	Mid.CBF MCS shall implement a set of tests to be executed each time the software is submitted in the software library (on push). The CI guidelines are provided on the SKA Developer Portal.

4.10 Software reliability requirements

Table 4-6 Software reliability requirements

ID	Requirements
Mid.CBF-MCS-PI3-REQ-100	Mid.CBF Master Control Software shall catch and re-throw all exceptions.

4.11 Software maintainability requirements

Table 4-7 Software maintanability requirements

ID	Requirements
Mid.CBF-MCS-PI3-REQ-110	Mid.CBF Master Control Software source code shall be well documented.

4.12 Software safety requirements

Software safety requirements have not been identified.

4.13 Software configuration and delivery requirements

Table 4-8 Software configuration and delivery requirements

ID	Requirements
Mid.CBF-MCS-PI3-REQ-130	Mid.CBF MCS source code shall be available as a project in the SKA GitHub.
Mid.CBF-MCS-PI3-REQ-131	Containerised image of the Mid.CBF MCS software shall be available on the Nexus Server, as specified on the SKA Developer Portal. http://developer.skatelescope.org/en/latest/

4.14 Data definition and database requirements

Table 4-9 Data definition and database requirements

ID	Requirements
Mid.CBF-MCS-PI3-REQ-140	Mid.CBF MCS shall provide the TANGO Database where all Mid.CBF TANGO devices register during initialization.

4.15 Human factors related requirements

Requirements related to human factors have not been identified.

4.16 Adaptation and installation requirements

No specific adaptation and installation requirements have been identified.

5 Assumptions

Table 5-1 Assumptions

ID	Category	Assumption	Retirement Date
1	Design Constraint	On a scan configuration with invalid parameters, Mid.CBF Subarray shall transition to observing state IDLE if its observing state was previously IDLE or READY if its observing state was previously READY.	End of PI#4

2	Design Constraint	Mid.CBF Subarray shall remain in observing state CONFIGURING until destination addresses have been provided for all configured fine channels. Additionally, if an invalid JSON object is received, Mid.CBF Subarray shall remain in observing state CONFIGURING.	End of PI#4

6 Software Design Overview

This chapter describes the design of the Mid.CBF MCS from static, dynamic, and behavioural points of view.

6.1 Software Static Architecture

The Mid.CBF MCS for the MVP is comprised of the following components:

- Mid.CBF Master
- 2 Mid.CBF Subarray
- 4 Mid.CBF VCC Capabilities
- 4 Mid.CBF FSP Capabilities

Each of these components is implemented by one or more of these TANGO Device Classes:

- `CbfMaster`, based on the `SKAMaster` class
- `CbfSubarray`, based on the `SKASubarray` class
- `SearchWindow`, based on the `SKACapability` class
- `Vcc`, based on the `SKACapability` class
- `VccBand1And2`, `VccBand3`, `VccBand4`, and `VccBand5`, all based on the `SKACapability` class
- `VccSearchWindow`, based on the `SKACapability` class
- `Fsp`, based on the `SKACapability` class
- `FspCorr`, `FspPss`, `FspPst`, `FspVlbi`, all based on the `SKACapability` class
- `FspCorrSubarray`, based on the `SKASubarray` class
- `FspPssSubarray`, based on the `SKASubarray` class

6.1.1 Mid.CBF Master

Mid.CBF Master functionality is implemented by a single instance of `CbfMaster` (FIGURE 6-1), the inheritance diagram of which is given in FIGURE 6-2.

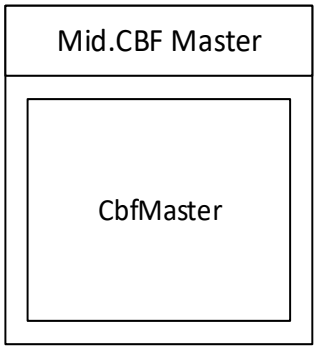

Figure 6-1 Composition of Mid.CBF Master

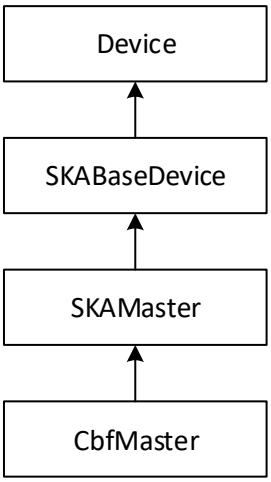

Figure 6-2 Inheritance diagram of Mid.CBF Master

6.1.2 Mid.CBF Subarray

Mid.CBF Subarray functionality is implemented by a single instance of `CbfSubarray` and two instances of `SearchWindow` (FIGURE 6-3). The inheritance diagram of these classes have been combined and is given in FIGURE 6-4.

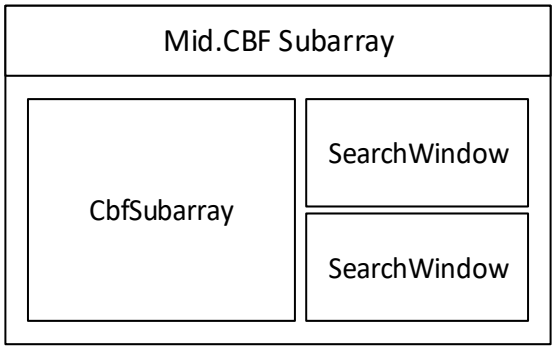

Figure 6-3 Composition of Mid.CBF Subarray

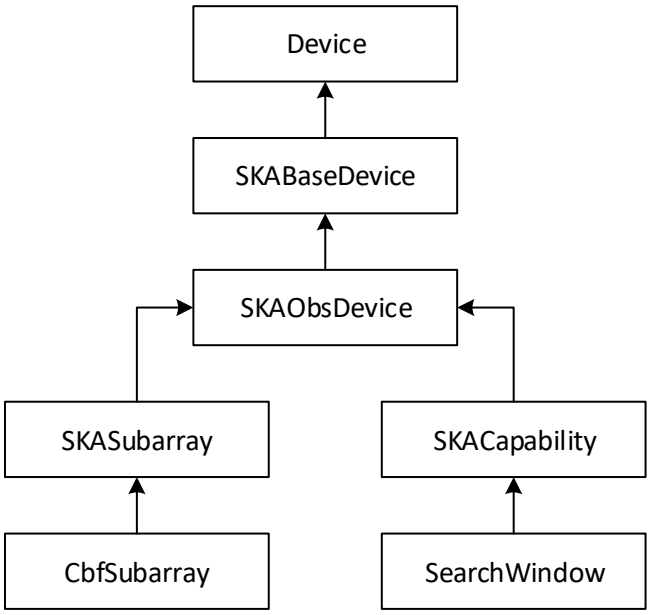

Figure 6-4 Inheritance diagram of Mid.CBF Subarray

6.1.3 Mid.CBF VCC Capability

Mid.CBF VCC Capability functionality is implemented by a single instance of `Vcc`, a single instance each of `VccBand1And2`, `VccBand3`, `VccBand4`, and `VccBand5`, and two instances of `VccSearchWindow` (FIGURE 6-5). The inheritance diagram of these classes have been combined and is given in FIGURE 6-6.

The TANGO Devices which implement the VCC's frequency band capabilities are instantiated during initialization. As described in **SECTION 7.3.6.3**, the VCC is configured to process input data for one frequency band at any given time. The active frequency band capability reports its operational state as ON, while all others report DISABLE.

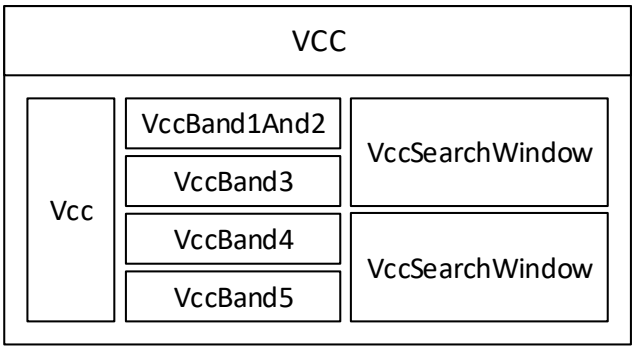

Figure 6-5 Composition of Mid.CBF VCC Capability

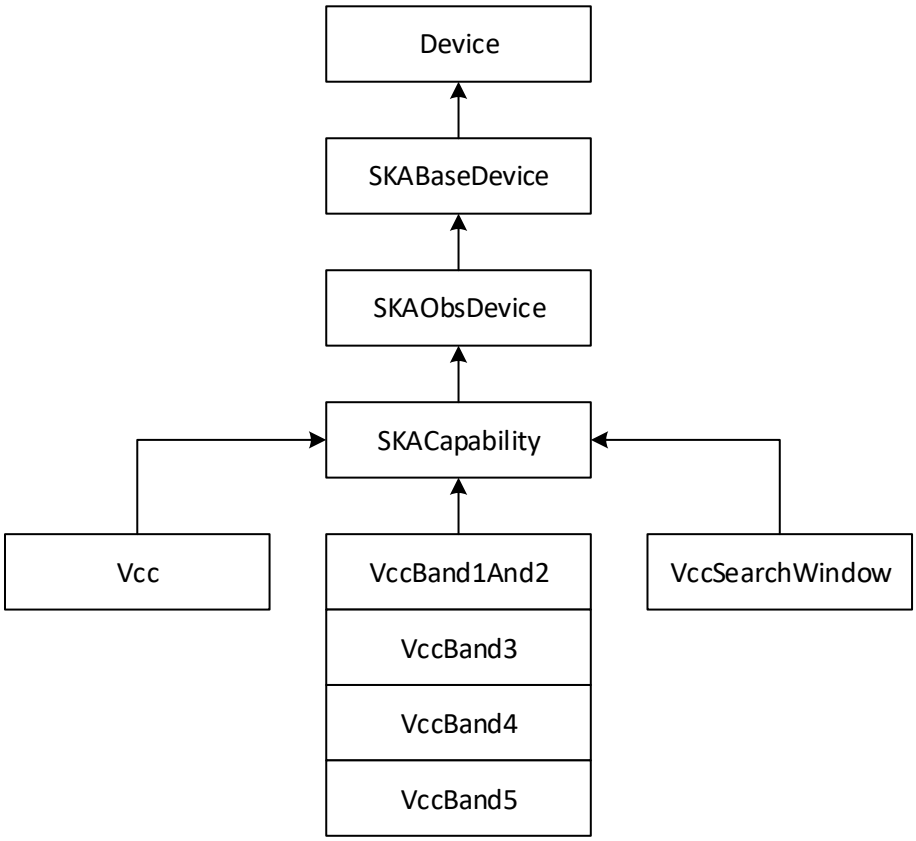

Figure 6-6 Inheritance diagram of Mid.CBF VCC Capability

6.1.4 Mid.CBF FSP Capability

Mid.CBF FSP Capability functionality is implemented by a single instance of `Fsp`, a single instance each of `FspCorr`, `FspPss`, `FspPst`, and `FspVlbi`, and a number of instances of `FspCorrSubarray`, `FspPssSubarray`, `FspVlbiSubarray`, and `fspPstSubarray` corresponding to

the number of Mid.CBF Subarrays (a single one for the MVP, future releases will instantiate up to 16 sub-arrays as required) (FIGURE 6-7). The inheritance diagram of these classes have been combined and is given in FIGURE 6-8.
The TANGO Devices which implement the FSP's function mode capabilities are instantiated during initialization. As described in SECTION 7.4.6.3, the FSP is configured to perform at most one function at any given time. The active function mode capability reports its operational state as ON, while all others report DISABLE.

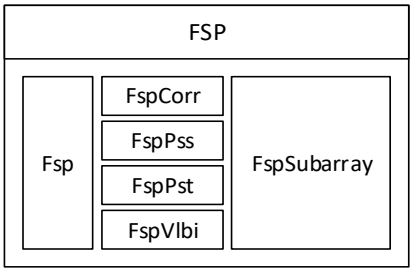

Figure 6-7 Composition of Mid.CBF FSP Capability

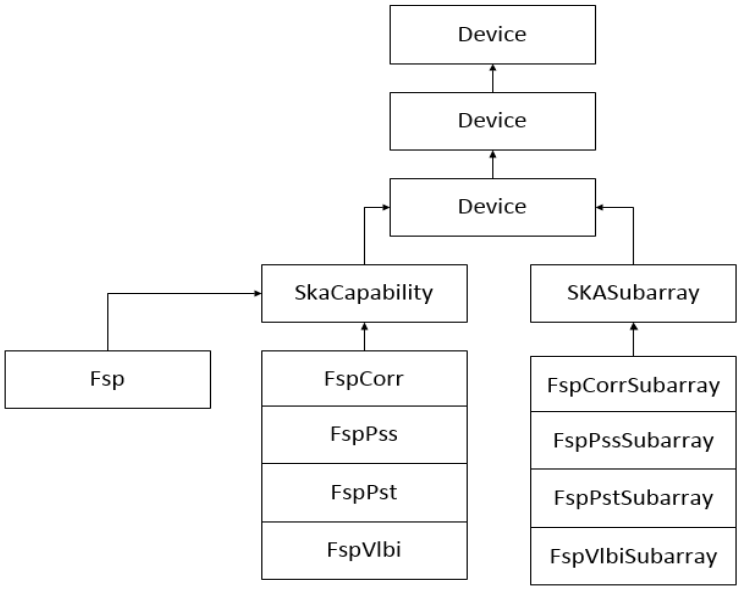

Figure 6-8 Inheritance diagram of Mid.CBF FSP Capability

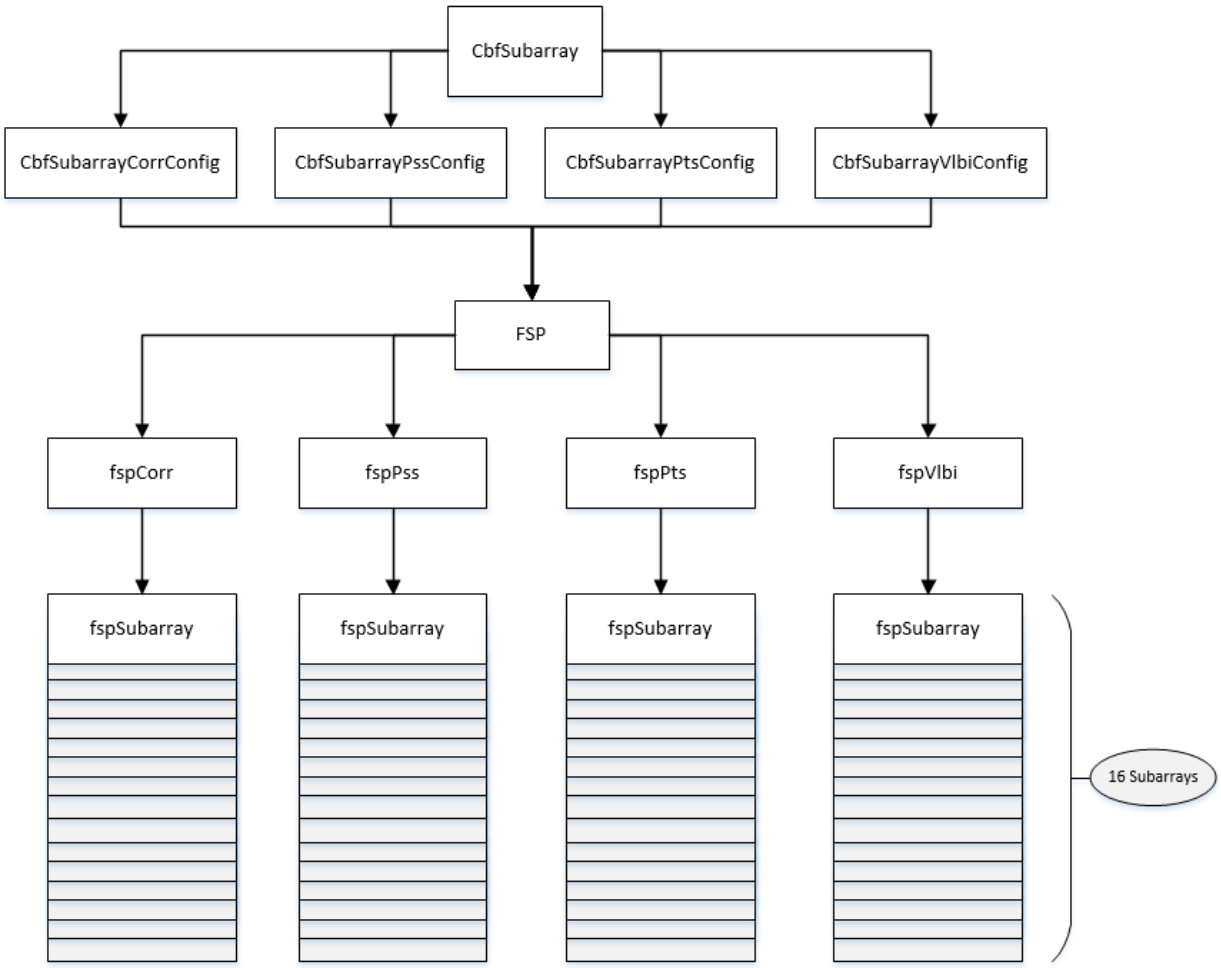

Figure 6-9 Inheritance diagram of Fsp's

6.2 Software Dynamic Architecture

The Mid.CBF MCS runs in a containerised environment (refer to CHAPTER 10 for more details). Each component runs in a separate TANGO Device Server inside its own container. In particular, the device servers implemented are:

- `CbfMaster`, for Mid.CBF Master, which runs the devices in FIGURE 6-1
- `CbfSubarrayMulti`, for the single instance of Mid.CBF Subarray, each of which runs the devices in FIGURE 6-3
- `VccMulti`, for the four instances of Mid.CBF VCC Capabilities, each of which runs the devices in FIGURE 6-5

- `FspMulti`, for the four instances of Mid.CBF FSP Capabilities, each of which runs the devices in FIGURE 6-7

Any `Multi` suffix indicates that the device server runs multiple device classes.
When the system is first started, the TANGO Devices are registered in a TANGO Database. For the MVP, integration and demo are done on the TMC side. Hence, all CSP devices (including the Mid.CBF devices listed in TABLE 6-1) register in the TMC TANGO DB. In the future, the CSP and Mid.CBF devices will register in a separate database. For development, the Mid.CBF devices will register locally in the MariaDB/MySQL database on the developer's machine.

6.2.1 Device FQDNs

The FQDNs of the components and sub-components of the Mid.CBF MCS are listed in TABLE 6-1.

Table 6-1 Mid.CBF MCS Device FQDNs

Device Class	FQDNs
CbfMaster	mid_csp_cbf/sub_elt/master
CbfSubarray	mid_csp_cbf/sub_elt/subarray_01
SearchWindow	mid_csp_cbf/sw1/subarray_01 mid_csp_cbf/sw2/subarray_01
Vcc	mid_csp_cbf/vcc/001 mid_csp_cbf/vcc/002 mid_csp_cbf/vcc/003 mid_csp_cbf/vcc/004
VccBand1And2	mid_csp_cbf/vcc_band12/001 mid_csp_cbf/vcc_band12/002 mid_csp_cbf/vcc_band12/003 mid_csp_cbf/vcc_band12/004
VccBand3	mid_csp_cbf/vcc_band3/001 mid_csp_cbf/vcc_band3/002 mid_csp_cbf/vcc_band3/003 mid_csp_cbf/vcc_band3/004
VccBand4	mid_csp_cbf/vcc_band4/001 mid_csp_cbf/vcc_band4/002 mid_csp_cbf/vcc_band4/003 mid_csp_cbf/vcc_band4/004
VccBand5	mid_csp_cbf/vcc_band5/001 mid_csp_cbf/vcc_band5/002 mid_csp_cbf/vcc_band5/003 mid_csp_cbf/vcc_band5/004

Device Class	FQDNs
VccSearchWindow	mid_csp_cbf/vcc_sw1/001 mid_csp_cbf/vcc_sw2/001 mid_csp_cbf/vcc_sw1/002 mid_csp_cbf/vcc_sw2/002 mid_csp_cbf/vcc_sw1/003 mid_csp_cbf/vcc_sw2/003 mid_csp_cbf/vcc_sw1/004 mid_csp_cbf/vcc_sw2/004
Fsp	mid_csp_cbf/fsp/01 mid_csp_cbf/fsp/02 mid_csp_cbf/fsp/03 mid_csp_cbf/fsp/04
FspCorr	mid_csp_cbf/fsp_corr/01 mid_csp_cbf/fsp_corr/02 mid_csp_cbf/fsp_corr/03 mid_csp_cbf/fsp_corr/04
FspPss	mid_csp_cbf/fsp_pss/01 mid_csp_cbf/fsp_pss/02 mid_csp_cbf/fsp_pss/03 mid_csp_cbf/fsp_pss/04
FspPst	mid_csp_cbf/fsp_pst/01 mid_csp_cbf/fsp_pst/02 mid_csp_cbf/fsp_pst/03 mid_csp_cbf/fsp_pst/04
FspVlbi	mid_csp_cbf/fsp_vlbi/01 mid_csp_cbf/fsp_vlbi/02 mid_csp_cbf/fsp_vlbi/03 mid_csp_cbf/fsp_vlbi/04
CbfSubarrayCorrConfig	mid_csp_cbf/corrConfig/01 mid_csp_cbf/corrConfig/02
CbfSubarrayPssConfig	mid_csp_cbf/pssConfig/01 mid_csp_cbf/pssConfig/02
CbfSubarrayPstConfig	mid_csp_cbf/pstConfig/01 mid_csp_cbf/pstConfig/02
CbfSubarrayVlbiConfig	mid_csp_cbf/vlbiConfig/01 mid_csp_cbf/vlbiConfig/02

Device Class	FQDNs
FspCorrSubarray	mid_csp_cbf/fspCorrSubarray/01_01 mid_csp_cbf/fspCorrSubarray/02_01 mid_csp_cbf/fspCorrSubarray/03_01 mid_csp_cbf/fspCorrSubarray/04_01
FspPssSubarray	mid_csp_cbf/fspPssSubarray/01_01 mid_csp_cbf/fspPssSubarray/02_01 mid_csp_cbf/fspPssSubarray/03_01 mid_csp_cbf/fspPssSubarray/04_01
FspPstSubarray	mid_csp_cbf/fspPstSubarray/01_01 mid_csp_cbf/fspPstSubarray/02_01 mid_csp_cbf/fspPstSubarray/03_01 mid_csp_cbf/fspPstSubarray/04_01
FspVlbiSubarray	mid_csp_cbf/fspVlbiSubarray/01_01 mid_csp_cbf/fspVlbiSubarray/02_01 mid_csp_cbf/fspVlbiSubarray/03_01 mid_csp_cbf/fspVlbiSubarray/04_01

6.2.2 Operational State and Observing State

The Mid.CBF MCS implements a set of operational states and observing states, reported as `state` and `obsState`, respectively, by relevant devices.

6.2.2.1 Mid.CBF Master

Mid.CBF Master implements the set of operational states given in FIGURE 6-9. It does not implement a set of observing states.

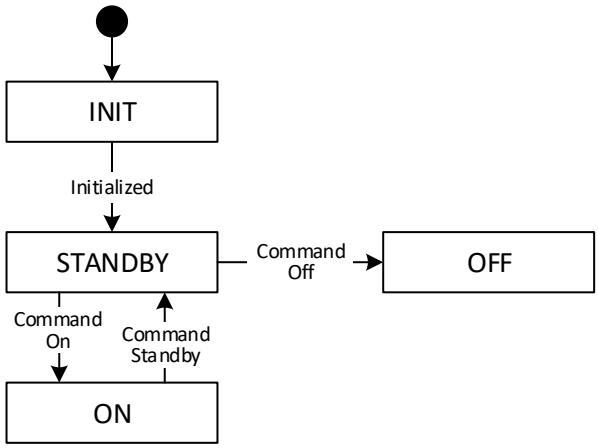

Figure 6-9 Operational states implemented by Mid.CBF Master

6.2.2.2 Mid.CBF Subarray

Mid.CBF Subarray state machine is revised according to ADR-8 in August 2020. Figure 6-10 describes the state machine for the base class "SKA Subarray". However, Mid.CBF subarray has several commands that has different names and input types.

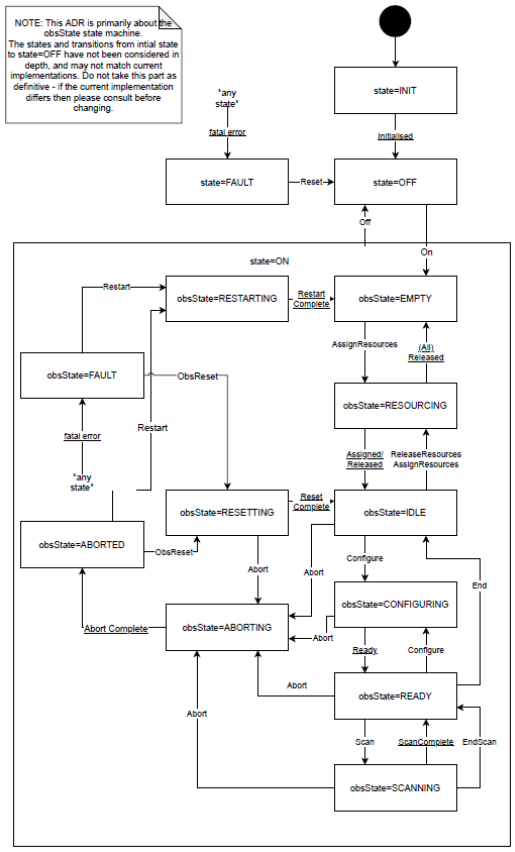

Figure 6-10 SKA Subarray State Machine Diagram

FIGURE 6-10 describes the SKA Subarray state machine. The upper part of the diagram describes the operational state. The lower part, contained in the box, describes the observing state.

The differences between SKA Subarray and Cbf Subarray commands are listed in TABLE 6-2

Table 6-2 different command name in Cbf Subarray

SKA Subarray	Cbf Subarray
End	GoToIdle
Configure	ConfigureScan
AssignRecources	AddReceptors
ReleaseRecources	RemoveReceptors() / RemoveAllReceptors()

6.2.2.3 Mid.CBF VCC Capability

Mid.CBF VCC Capability implements the set of operational states given in FIGURE 6-11 and the set of observing states given in FIGURE 6-12.

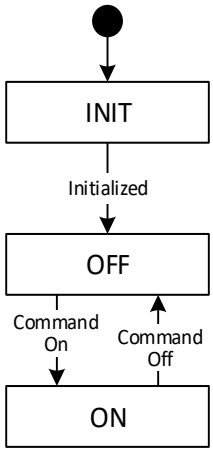

Figure 6-11 Operational states implemented by Mid.CBF VCC Capability

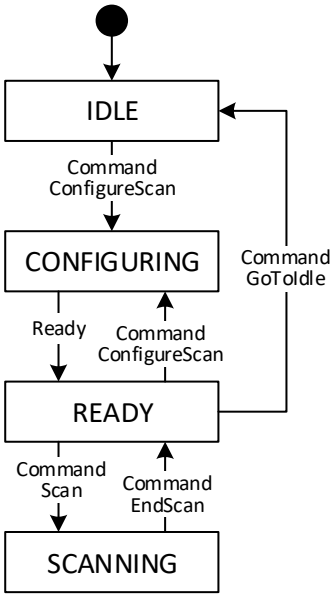

Figure 6-12 Observing states implemented by Mid.CBF VCC Capability

6.2.2.4 Mid.CBF FSP Capability

Mid.CBF FSP Capability implements the set of operational states given in FIGURE 6-13. It does not implement a set of observing states.

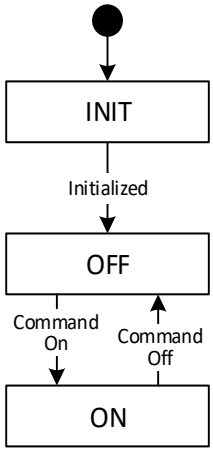

Figure 6-13 Operational states implemented by Mid.CBF FSP Capability

6.2.2.5 VCC Frequency Band and Search Window and FSP Function Mode Capabilities

The Mid.CBF VCC frequency band and search window and FSP function mode Capabilities all implement the set of operational states given in FIGURE 6-14. They do not implement a set of observing states.

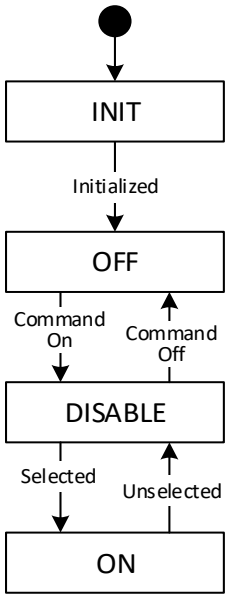

Figure 6-14 Operational states implemented by Mid.CBF VCC frequency band and search window and FSP function mode Capabilities

6.2.2.6 Mid.CBF FSP Subarray Capability

Mid.CBF FSP Subarray Capability implements the set of observing states given in FIGURE 6-15. Its operational state takes on the operational state of the FSP it belongs to.

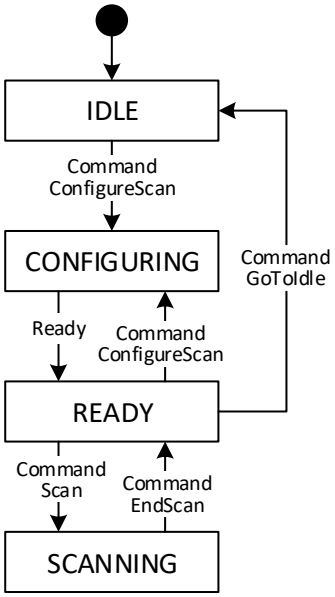

Figure 6-15 Observing states implemented by Mid.CBF FSP Subarray Capability

6.2.3 State Transitions

TABLE 6- provides an outline of a typical workflow that Mid.CBF may execute, along with the resulting state transitions for the devices implemented for the MVP (a single Mid.CBF Subarray, 4 VCCs, and 4 FSPs). A set of states is preceded by a trigger (darker grey background), and the important transitions thereafter are **bolded**.

Table 6-3 Mid.CBF MCS state transitions

The Mid.CBF first receives power. All devices enter an initializing state.						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
INIT	**INIT**	EMPTY	**INIT**	IDLE	**INIT**	IDLE
			INIT	IDLE	**INIT**	IDLE
			INIT	IDLE	**INIT**	IDLE
			INIT	IDLE	**INIT**	IDLE
The Mid.CBF is finished initializing and enters a low-power standby state. The Mid.CBF Subarrays are disabled, and the VCC and FSP Capabilities are powered off.						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
STANDBY	**OFF**	EMPTY	**OFF**	IDLE	**OFF**	IDLE

			OFF	IDLE	**OFF**	IDLE
			OFF	IDLE	**OFF**	IDLE
			OFF	IDLE	**OFF**	IDLE
The command to power on is sent to Mid.CBF. The Mid.CBF Subarrays are powered on, and the VCC and FSP Capabilities are powered on.						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
ON	**ON**	EMPTY	**ON**	IDLE	**ON**	IDLE
			ON	IDLE	**ON**	IDLE
			ON	IDLE	**ON**	IDLE
			ON	IDLE	**ON**	IDLE
Receptors are added to the Mid.CBF Subarray, which subsequently enters the IDLE obsState						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
ON	ON	**IDLE**	ON	IDLE	ON	IDLE
			ON	IDLE	ON	IDLE
			ON	IDLE	ON	IDLE
			ON	IDLE	ON	IDLE
The command to configure a scan is sent to the Mid.CBF Subarray, which subsequently enters observing state CONFIGURING. Additionally, all the FSP Subarrays(not FSP, but FSP Subarray) added to the scan and all the VCCs added in the previous step enter observing state CONFIGURING. In this example, 4 VCCs (in the previous step) and 2 FSPs were added.						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
ON	ON	**CONFIGURING**	ON	**CONFIGURING**	ON	**CONFIGURING**
			ON	**CONFIGURING**	ON	**CONFIGURING**
			ON	**CONFIGURING**	ON	IDLE
			ON	**CONFIGURING**	ON	IDLE
The Mid.CBF Subarray is finished configuring the scan. All configured observing devices enter observing state READY.						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
ON	ON	**READY**	ON	**READY**	ON	**READY**
			ON	**READY**	ON	**READY**
			ON	**READY**	ON	IDLE
			ON	**READY**	ON	IDLE
The command to start the scan(“Scan”) is sent to the Mid.CBF Subarray. All configured observing devices enter observing state SCANNING.						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
ON	ON	**SCANNING**	ON	**SCANNING**	ON	**SCANNING**
			ON	**SCANNING**	ON	**SCANNING**
			ON	**SCANNING**	ON	IDLE
			ON	**SCANNING**	ON	IDLE

The command to end the scan("EndScan") is sent to the Mid.CBF Subarray. All configured observing devices enter observing state READY.						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
ON	ON	**READY**	ON	**READY**	ON	**READY**
			ON	**READY**	ON	**READY**
			ON	**READY**	ON	IDLE
			ON	**READY**	ON	IDLE
The command "GoToIdle" is sent to the Mid.CBF Subarray, which subsequently enters observing state IDLE. All configured VCCs and FSP Subarrays enter observing state IDLE. The FSPs are released from the sub-array, but the VCCs are remain affiliated.						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
ON	ON	**IDLE**	ON	**IDLE**	ON	**IDLE**
			ON	**IDLE**	ON	**IDLE**
			ON	**IDLE**	ON	IDLE
			ON	**IDLE**	ON	IDLE
All receptors are removed from the Mid.CBF Subarray, which subsequently turns observing state to EMPTY						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
ON	ON	**EMPTY**	ON	IDLE	ON	IDLE
			ON	IDLE	ON	IDLE
			ON	IDLE	ON	IDLE
			ON	IDLE	ON	IDLE
The command to enter the low-power standby state is sent to Mid.CBF. The Mid.CBF Subarrays are disabled, and the VCC and FSP Capabilities are powered off. (to trigger this in JIVE, send STANBY command to CBF Master)						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
STANDBY	**OFF**	IDLE	**OFF**	IDLE	**OFF**	IDLE
			OFF	IDLE	**OFF**	IDLE
			OFF	IDLE	**OFF**	IDLE
			OFF	IDLE	**OFF**	IDLE
The command to power off is sent to Mid.CBF. (Send OFF command to CBF Master)						
Mid.CBF state	Mid.CBF Subarray state	Mid.CBF Subarray obsState	VCC state	VCC obsState	FSP state	FSP Subarray obsState
OFF	OFF	IDLE	OFF	IDLE	OFF	IDLE
			OFF	IDLE	OFF	IDLE
			OFF	IDLE	OFF	IDLE
			OFF	IDLE	OFF	IDLE

6.3 Software Behaviour

6.3.1 Power On, Off, and Standby

On startup, after initialization, Mid.CBF transitions to a low-power STANDBY state, drawing <5% of nominal power. Consequently, all Mid.CBF Subarrays are in DISABLE state, and all VCC and FSP Capabilities are OFF. All relevant observing states are IDLE.
The following sequence of messages, shown in FIGURE 6-16, occurs when powering on Mid.CBF, which is only allowed when Mid.CBF is in STANDBY state:

1. TMC CSP Master Leaf Node command `On` is issued.
2. TMC CSP Master Leaf Node invokes CSP Master command `On`.
3. CSP Master subsequently invokes Mid.CBF Master command `On`.
4. Mid.CBF Master subsequently invokes the `On` command of all Mid.CBF Subarrays (transitioning them to OFF state) and VCC and FSP Capabilities (transitioning them to ON state). Once the hardware is deployed, Mid.CBF Master will turn on the VCCs and FSPs gradually, to keep inrush current within required limits. This release does not implement this functionality.
5. Mid.CBF Master transitions to ON state.
6. CSP Master subsequently transitions to ON state.

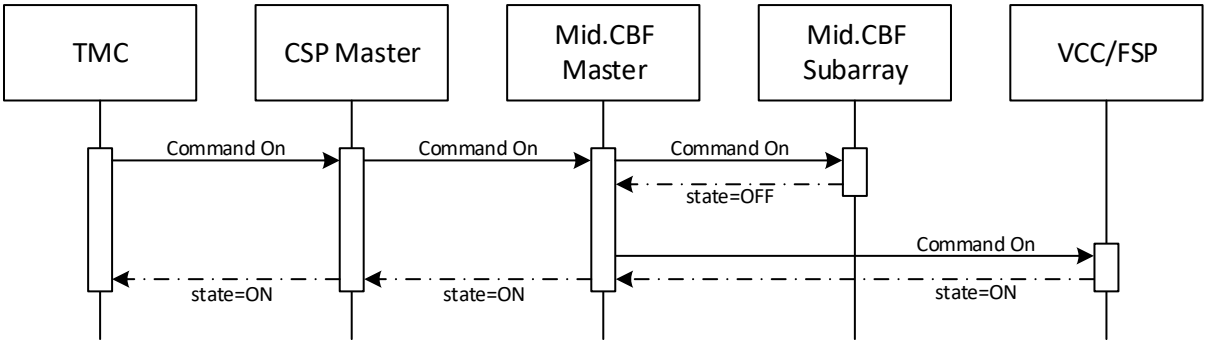

Figure 6-16 Message flow when powering on Mid.CBF

The sequence of messages, shown in FIGURE 6-17, that occurs when the Mid.CBF is put into STANDBY state, which is only allowed when Mid.CBF is in ON state, is similar and as follows:

1. TMC CSP Master Leaf Node command `Standby` is issued.
2. TMC CSP Master Leaf Node invokes CSP Master command `Standby`.
3. CSP Master subsequently invokes Mid.CBF Master command `Standby`.
4. Mid.CBF Master subsequently invokes the `Off` command of all Mid.CBF Subarrays (transitioning them to DISABLE state) and VCC and FSP Capabilities (transitioning them to OFF state).

5. Mid.CBF Master transitions to STANDBY state.
6. CSP Master subsequently transitions to STANDBY state.

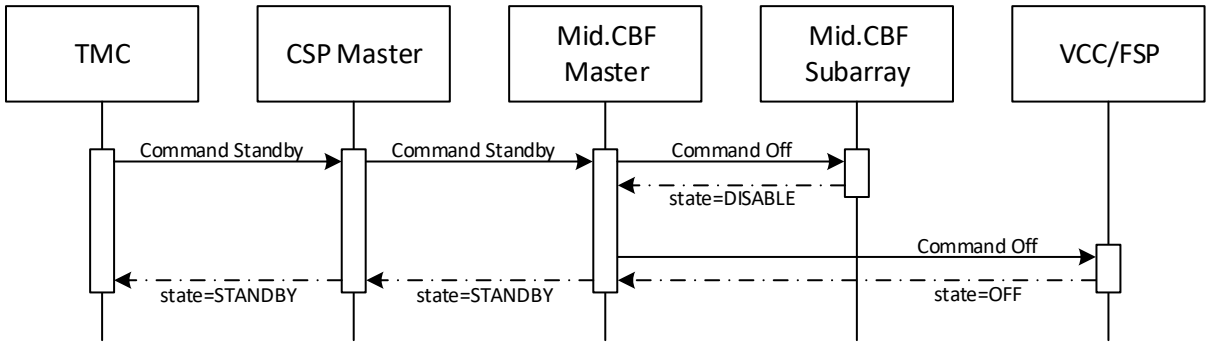

Figure 6-17 Message flow when putting Mid.CBF into standby

The following sequence of messages, shown in FIGURE 6-18, occurs when powering off Mid.CBF, which is only allowed when Mid.CBF is already in STANDBY state:

1. TMC CSP Master Leaf Node command `Off` is issued.
2. TMC CSP Master Leaf Node invokes CSP Master command `Off`.
3. CSP Master subsequently invokes Mid.CBF Master command `Off`.
4. Mid.CBF Master invokes the `Off` command of all Mid.CBF Subarrays (transitioning them to DISABLE state) and VCC and FSP Capabilities (transitioning them to OFF state). Even though this command is allowed only in STANDBY state, when the VCCs and FSPs are already OFF, Mid.CBF Master still forwards the `Off` command. The complete implementation may include powering off the network switches, power supplies, and other equipment.
5. Mid.CBF Master transitions to OFF state.
6. CSP Master subsequently transitions to OFF state.

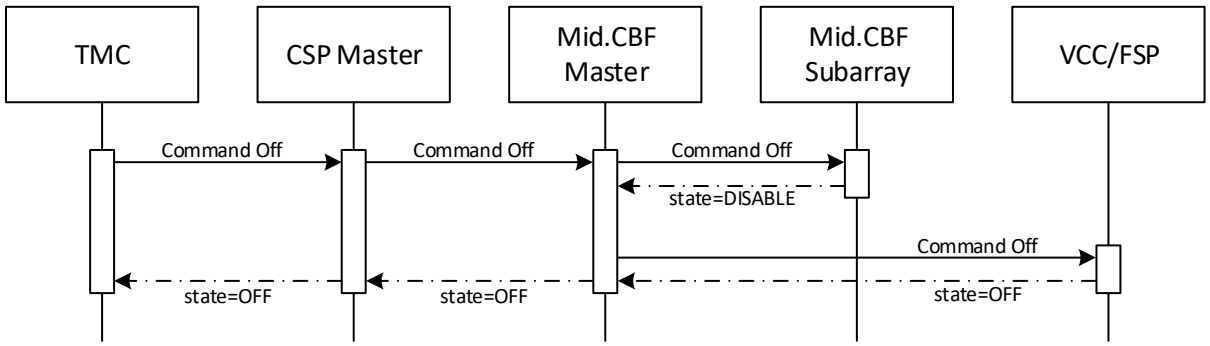

Figure 6-18 Message flow when powering off Mid.CBF

6.3.2 Assign and Release Resources

Receptors are assigned and released to and from a particular Mid.CBF Subarray in advance of scan configuration, and is only allowed in observing state IDLE. The sequences of messages, shown in FIGURE 6-19, is as follows:

1. The TMC CSP Subarray Leaf Node command `AssignResources` is issued with a list of resources to assign to the Mid.CBF Subarray, which is mapped to receptors.
2. The TMC CSP Subarray Leaf Node invokes the CSP Subarray command `AddReceptors`.
3. The CSP Subarray subsequently invokes the Mid.CBF Subarray command `AddReceptors`.
4. The Mid.CBF Subarray changes the sub-array affiliation of each assigned receptor's corresponding VCC, assigning it to itself and preventing it from being assigned to another sub-array (this command fails if the corresponding VCC of a receptor being assigned is already affiliated with a different sub-array). Additionally, the Mid.CBF Subarray subscribes to receive change event notifications for each assigned VCC's `state` and `healthState` attributes.
5. The Mid.CBF Subarray transitions to ON state.
6. The CSP Subarray subsequently transitions to ON state.

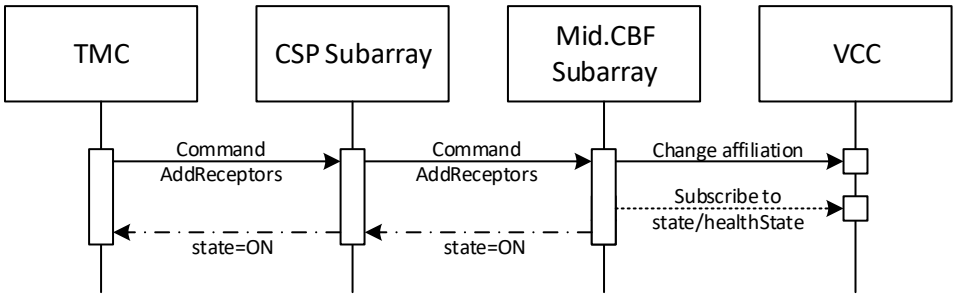

Figure 6-19 Message flow when assigning resources to a Mid.CBF Subarray

Resources are released from a particular Mid.CBF Subarray in a similar fashion. For the MVP, it is not possible for the TMC to selectively release resources assigned to Mid.CBF (it is only possible to release all assigned resources at once). The sequences of messages, shown in FIGURE 6-20, is as follows:

1. The TMC CSP Subarray Leaf Node command `RemoveAllResources` is issued.
2. The TMC CSP Subarray Leaf Node invokes the CSP Subarray command `RemoveAllReceptors`.
3. The CSP Subarray subsequently invokes the Mid.CBF Subarray command `RemoveAllReceptors`.
4. The Mid.CBF Subarray resets the sub-array affiliation of each released receptor's corresponding `Vcc` and unsubscribes from any event notifications.
5. The Mid.CBF Subarray transitions to OFF state.

6. The CSP Subarray subsequently transitions to OFF state.

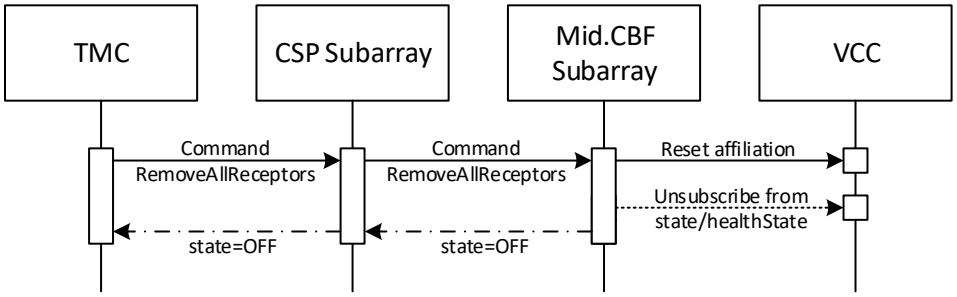

Figure 6-20 Message flow when releasing all resources from a Mid.CBF Subarray

6.3.3 Scan Configuration

In order to configure a scan, a Mid.CBF Subarray must be in the ON state. Consequently, at least one receptor must be assigned to the Mid.CBF Subarray.
The scan configuration parameters are passed in the form of a string containing a serialized JSON object. The sequences of messages, shown in FIGURE 6-21, is as follows:

1. The TMC CSP Subarray Leaf Node command `ConfigureScan` is issued with the scan configuration parameters.
2. The TMC CSP Subarray Leaf Node invokes the CSP Subarray command `ConfigureScan`.
3. The CSP Subarray subsequently invokes the Mid.CBF Subarray command `ConfigureScan`.
4. The Mid.CBF Subarray transitions to observing state CONFIGURING.
5. The CSP Subarray subsequently transitions to observing state CONFIGURING.
6. The Mid.CBF Subarray configures its affiliated VCCs by setting relevant attributes, including their observed frequency band, and configuring search windows if specified in the scan configuration. The VCCs then transition to observing state READY.
7. The Mid.CBF Subarray configures FSPs, including setting their function mode, destination addresses, output links, and subscribes to receive change event notifications for their `state` and `healthState` attributes. The FSP is now affiliated with the Mid.CBF Subarray (in addition to any affiliations it may have previously had).
8. CbfSubarrayConfig classes store the most recent scan configuration and then send the config to the respective fspCorrSubarry, fspPssSubarray, fspVlbiSubarray, or fspPstSubarray depending on the function mode of the scan.
9. When the FSP Subarrays receive the destination addresses for all fine channels, they transition to observing state READY. Subsequently, the Mid.CBF Subarray transitions to observing state READY.

10. The CSP Subarray subsequently transitions to observing state READY.
11. The TMC possibly publishes delay models, which is received by the Mid.CBF Subarray and forwarded to its affiliated VCCs.

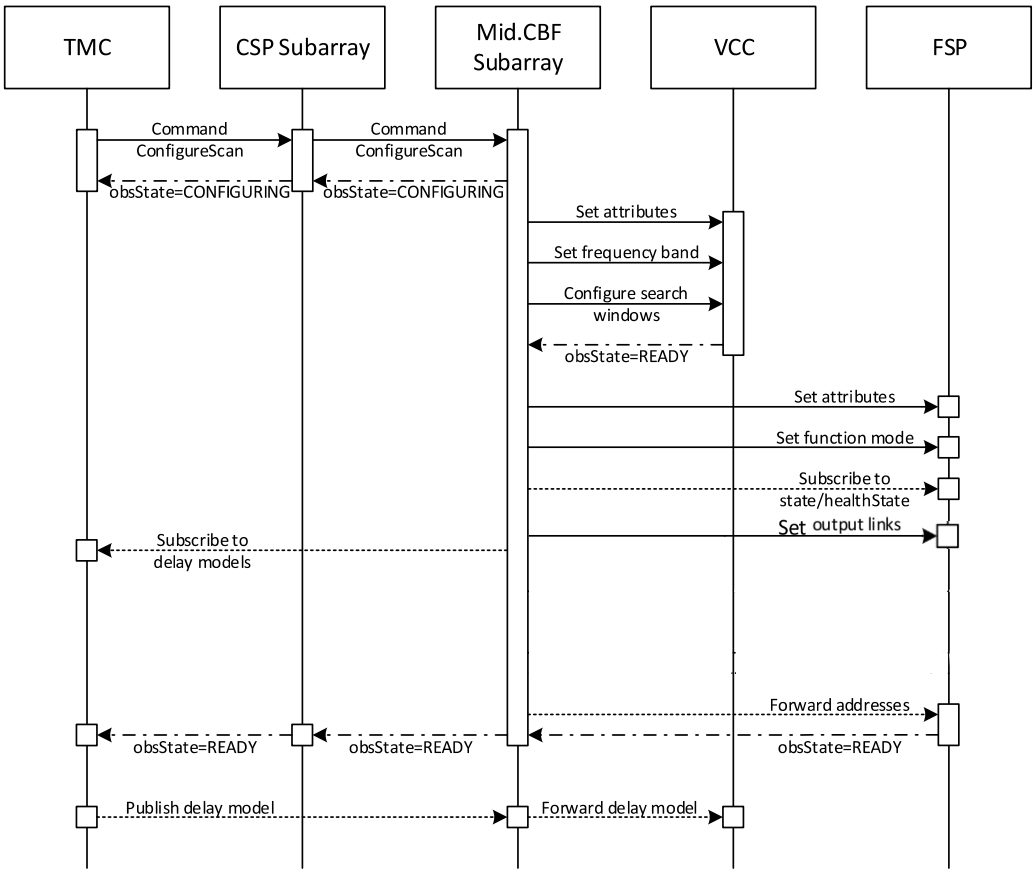

Figure 6-21 Message flow when configuring a scan

6.3.4 Scan Execution

A scan can only be started when the Mid.CBF Subarray is in observing state READY. Consequently, the Mid.CBF Subarray must have received the SDP destination addresses. However, it is not necessary for the delay models to be updated before starting a scan. The following sequence of messages, shown in FIGURE 6-22, occurs when performing a scan:

1. The TMC CSP Subarray Leaf Node command `StartScan` is issued.
2. The TMC CSP Subarray Leaf Node invokes the CSP Subarray command `Scan`.
3. The CSP Subarray subsequently invokes the Mid.CBF Subarray command `Scan`.

4. The Mid.CBF Subarray subsequently invokes the Scan command of its affiliated VCCs and FSP Subarrays, transitioning them to observing state SCANNING.
5. The Mid.CBF Subarray transitions to observing state SCANNING.
6. The CSP Subarray subsequently transitions to observing state SCANNING.
7. The TMC publishes delay model updates, which is received by the Mid.CBF Subarray and forwarded to its affiliated VCCs.
8. The TMC CSP Subarray Leaf Node command EndScan is issued.
9. The TMC CSP Subarray Leaf Node invokes the CSP Subarray command EndScan.
10. The CSP Subarray subsequently invokes the Mid.CBF Subarray command EndScan.
11. The Mid.CBF Subarray subsequently invokes the EndScan command of its affiliated VCCs and FSPs, transitioning them to observing state READY.
12. The Mid.CBF Subarray transitions to observing state READY.
13. The CSP Subarray subsequently transitions to observing state READY.

Delay models should be published before the Mid.CBF Subarray transitions to observing state SCANNING, and should be periodically updated during scan execution. If the delay models are not received before Mid.CBF transitions to SCANNING, Mid.CBF will start generating output products but will flag the products as invalid. Similarly, if a delay model of one or more receptors expires during scan execution, Mid.CBF will continue to generate output products, but will flag them as invalid. As soon as the delay models are updated, Mid.CBF will resume normal operation and the products will be marked valid. (Since output products are not generated for the MVP, none of this functionality has been implemented yet.)

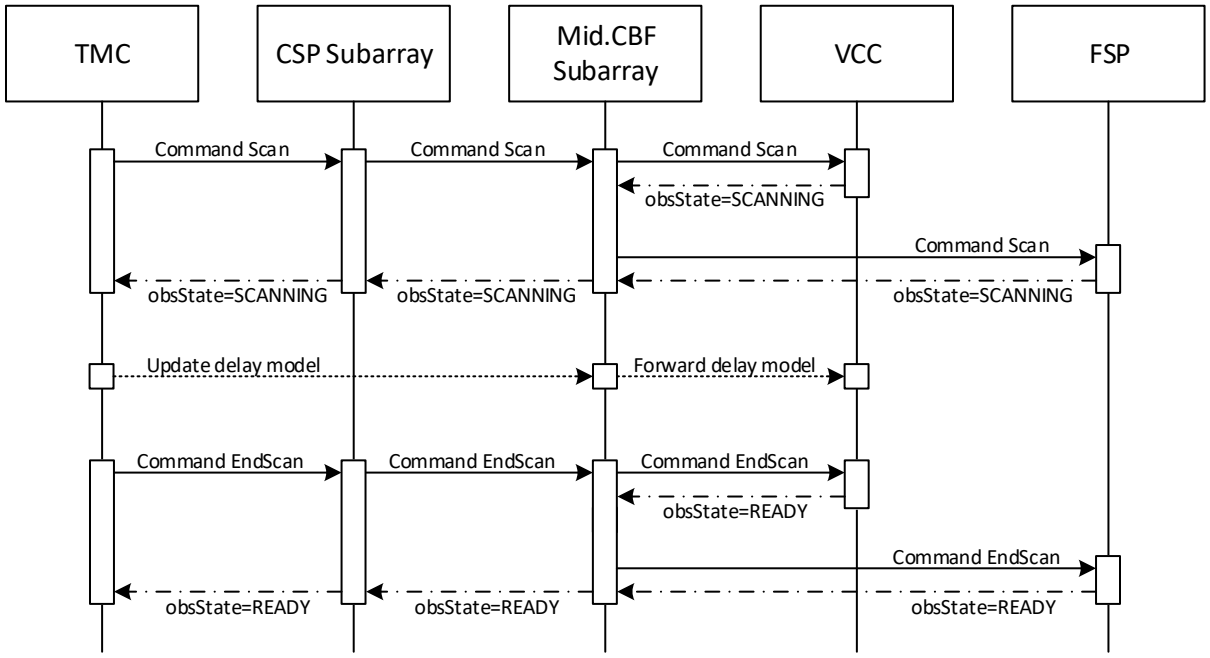

Figure 6-22 Message flow when performing a scan

6.3.5 End Scheduling Block[2]

When a scan has been configured, it may be deconfigured and the resources (excluding any VCCs) released by ending the scheduling block. However, the scheduling block may not be ended during a scan – that is, when the Mid.CBF Subarray is in observing state SCANNING. The following sequence of messages, shown in FIGURE 6-23, occurs when performing a scan:

1. The TMC CSP Subarray Leaf Node command `EndSB` is issued.
2. The TMC CSP Subarray Leaf Node invokes the CSP Subarray command `EndSB`.
3. The CSP Subarray subsequently invokes the Mid.CBF Subarray command `EndSB`.
4. The Mid.CBF Subarray unsubscribes from any events whose subscription points were given during scan configuration, including updates to the delay model provided by the TMC and destination addresses provided by the SDP.
5. The Mid.CBF Subarray unsubscribes from any FSP `state` and `healthState` event notifications.
6. The Mid.CBF Subarray invokes the `GoToIdle` command of its affiliated VCCs and FSPs (after which the FSPs are no longer affiliated).
7. The Mid.CBF Subarray transitions to observing state IDLE.
8. The CSP Subarray subsequently transitions to observing state IDLE.

[2] This command is called “EndSB” at the moment, but will be changed to “GoToIdle”. Mid.CBF Subarray should not be aware of scheduling blocks.

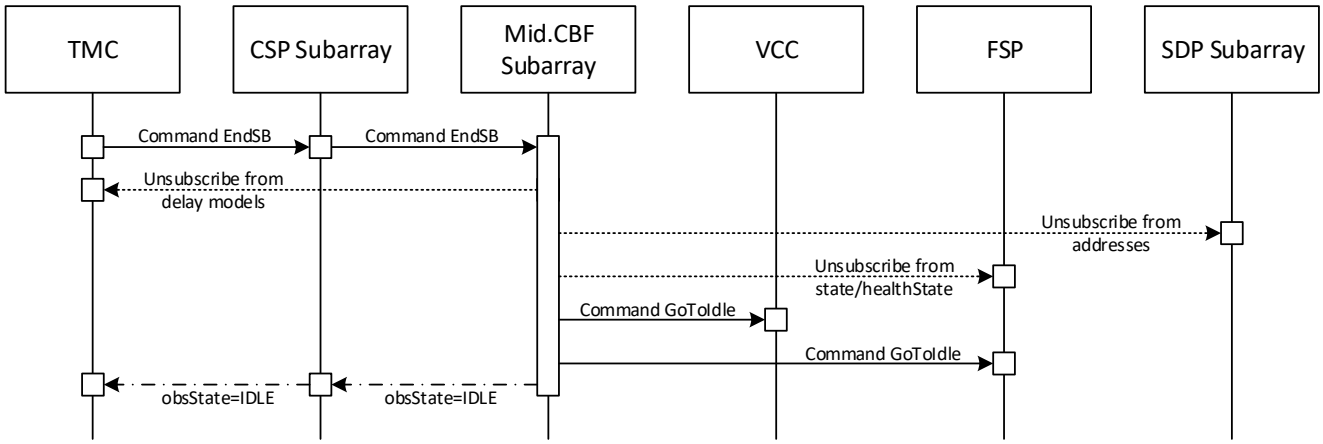

Figure 6-23 Message flow when ending a scheduling block

6.4 Error Handling

In general, during command execution on Mid.CBF Master or a Mid.CBF Subarray, any exceptions thrown by a VCC or FSP Capability will be caught, logged, and re-thrown. The only exceptions to this rule are the state change commands `On`, `Off` and `Standby`, where the propagated command will fail silently.
Additionally, all dynamic event subscriptions will raise an exception if the subscription fails.

6.5 Monitor and Control

This section is not applicable to the Mid.CBF MCS; the purpose of the MCS itself is to monitor and control the TALON-DX boards that implement signal processing functionality.

6.6 External Interfaces

Mid.CBF Master and Mid.CBF Subarray gives provision for CSP.LMC (or any other TANGO Device) to access attributes and send commands. These are detailed in SECTION 7.1 (Mid.CBF Master) and SECTION 7.2 (Mid.CBF Subarray).
Though not implemented for the MVP, the Mid.CBF MCS will eventually interface with TALON-DX HPS devices.
Context diagrams are given in SECTION 3.3.

6.7 Long Lifetime Software

Since the entirety of the Mid.CBF MCS runs in a virtual machine inside Docker containers (refer to CHAPTER 10 for details), there is minimal dependency on the host operating system and hardware, thus improving portability.

6.8 Memory and CPU Budget

The memory and CPU budget of the Mid.CBF MCS has not yet been documented.

6.9 Design Standards, Conventions and Procedures

Programming standards adhere to the PEP 8 style guide for Python programming, in as large of a capacity as possible that does not conflict with TANGO requirements and automatically-generated code.
The names of the devices of the Mid.CBF MCS adhere to standard TANGO naming conventions and the SKA1 TANGO naming conventions described in AD3.
Efforts will be made in the future to improve in-code documentation.

6.10 Environment

Refer to CHAPTER 10 for environment details.

6.11 Reliability, Availability and Maintainability

The Mid.CBF MCS implements the following features to improve reliability and robustness:

- State checks in every applicable device when executing a command
- Redundant operations (e.g. clearing a data structure), possibly increasing overhead
- Validation of input arguments in deep commands (e.g. `ConfigureScan`) without side effects

No special considerations were given to the availability or maintainability of the Mid.CBF MCS during the design process.

7 Software Components

This chapter describes in detail the components of the Mid.CBF MCS identified in CHAPTER 6.

7.1 Mid.CBF Master

Mid.CBF Master registers in the TANGO database with the following FQDN:

`mid_csp_cbf/sub_elt/master`

7.1.1 Type

Mid.CBF Master is a TANGO Device Server implemented by the device class `CbfMaster`.

7.1.2 Development Type

Mid.CBF Master is newly-developed, open-source software.

7.1.3 Function and Purpose

Mid.CBF Master represents a primary point of contact for monitoring and control of Mid.CBF. It implements state and mode indicators for all Mid.CBF constituents and a set of state transition commands.

7.1.4 Subordinates

Mid.CBF Master has no immediate subordinates.

7.1.5 Dependencies

In the MVP, Mid.CBF Master, on startup, randomly assigns receptors IDs to the VCCs. For this reason, the VCC Capabilities must be fully initialized and running when the Mid.CBF Master initializes. This dependency will be removed in the future, when the link between a VCC and its correponding receptor is more permanent.
Commands that execute on other devices, namely the commands to transition state, do not explicitly require the presence of those devices. Any device that is unreachable will simply be skipped.

7.1.6 Interfaces

Mid.CBF Master implements a set of properties, attributes, and commands as an interface feature.

7.1.6.1 Properties

Mid.CBF Master implements the set of properties listed in TABLE 7-1.

Table 7-1 Mid.CBF Master Properties

Name	Type	Description
MaxCapabilities	`(str,)`	A map of the maximum number of available capabilities for each capability type. Each element has the form “<capability type>:<maximum number>”. The capability types Subarray, VCC, and FSP must be present. For example, a valid value of this property is (“Subarray:1”, “VCC:4”, “FSP:4”)
CbfSubarray	`(str,)`	An array of the FQDNs of all the Mid.CBF Subarrays.
VCC	`(str,)`	An array of the FQDNs of all the Mid.CBF VCC Capabilities.
FSP	`(str,)`	An array of the FQDNs of all the Mid.CBF FSP Capabilities.

7.1.6.2 Attributes

Mid.CBF Master implements the set of attributes listed in TABLE 7-2.

Table 7-2 Mid.CBF Master Attributes

Name	Type	Access	Description
receptorToVcc	`(str,)`	R/O	A map of the corresponding VCC for each receptor. Each element has the form “<receptor ID>:<VCC ID>”.
vcctoReceptor	`(str,)`	R/O	A map of the corresponding receptor for each VCC. Each element has the form “<VCC ID>:<receptor ID>”.
subarrayConfigID	`(str,)`	R/O	An array of the IDs of the configurations in each sub-array. (empty string if nothing is configured).
reportVCCState	`(DevState,)`	R/O	An array of the states of each VCC.

Name	Type	Access	Description
reportVCCHealthState	(uint16,)	R/O	An array of the health states of each VCC.
reportVCCAdminMode	(uint16,)	R/O	An array of the admin modes of each VCC.
reportVCCSubarrayMembership	(uint16,)	R/O	An array of the sub-array affiliations of each VCC.
reportFSPState	(DevState,)	R/O	An array of the states of each FSP.
reportFSPHealthState	(uint16,)	R/O	An array of the health states of each FSP.
reportFSPAdminMode	(uint16,)	R/O	An array of the admin modes of each FSP.
reportFSPSubarrayMembership	((uint16,),)	R/O	An array of the sub-array affiliations of each FSP. Each FSP can be affiliated with up to 16 sub-arrays simultaneously.
reportSubarrayState	(DevState,)	R/O	An array of the states of each Mid.CBF Subarray.
reportSubarrayHealthState	(uint16,)	R/O	An array of the health states of each Mid.CBF Subarray.
reportSubarrayAdminMode	(uint16,)	R/O	An array of the admin modes of each Mid.CBF Subarray.

7.1.6.3 Commands

Mid.CBF Master implements the set of commands listed in TABLE 7-3.

Table 7-3 Mid.CBF Master Commands

Name	Input Type	Output Type	Description
On	None	None	Turn on Mid.CBF, including all the Mid.CBF Subarrays and Mid.CBF Capabilities. Allowed only when the state of Mid.CBF Master is STANDBY.
Off	None	None	Turn off Mid.CBF, including all the Mid.CBF Subarrays and Mid.CBF Capabilities. Allowed only when the state of Mid.CBF Master is STANDBY.
Standby	None	None	Standby Mid.CBF, including all the Mid.CBF Subarrays and Mid.CBF Capabilities. Allowed only when the state of Mid.CBF Master is ON.

7.1.7 Resources

Mid.CBF Master requires no particular resources to operate.

7.1.8 References

AD1 and AD2 provide more details on the design and interfaces of Mid.CBF Master.

7.1.9 Data

All important and relevant data of Mid.CBF Master has been exposed as attributes (SECTION 7.1.6.2).

7.2 Mid.CBF Subarray

Mid.CBF Subarray registers in the TANGO database with the following FQDN:

`mid_csp_cbf/sub_elt/subarray_xx`

where `xx` refers to instances `01` through `16` (though only `01` is implemented as part of the MVP).

7.2.1 Type

Mid.CBF Subarray is a TANGO Device Server implemented predominantly by the device class `CbfSubarray` (though subordinate devices exist, see SECTION 7.2.4).

7.2.2 Development Type

Mid.CBF Subarray is newly-developed, open-source software.

7.2.3 Function and Purpose

Mid.CBF Subarray implements commands and attributes needed for scan configuration and execution. In particular, Mid.CBF Subarray gives provision for CSP.LMC (or any other TANGO Device) to perform the following:

- Add and release resources to and from a particular sub-array
- Configure a scan for imaging
- Perform a scan
- Deconfigure a scan

7.2.4 Subordinates

Mid.CBF Subarray has two subordinate devices, which are both instantiations of the `SearchWindow` device class. They implement attributes that specify parameters of the two search windows that can be configured for a scan. They register in the TANGO database with the following FQDNs:

```
mid_csp_cbf/sw1/xx
mid_csp_cbf/sw2/xx
```

where `xx` refers to instances `01` through `16` (though only `01` is implemented as part of the MVP).

7.2.5 Dependencies

Mid.CBF Subarray has no dependencies on startup. It may be initialized before or after the VCC and FSP Capabilities or Mid.CBF Master. However, the following dependencies exist for commands:

- Adding and removing receptors requires the presence of Mid.CBF Master and the receptors' corresponding VCCs that are being added or removed.
- Configuring a scan requires the presence of Mid.CBF Master, the VCCs that were assigned to the sub-array, and the FSPs specified in the scan parameters. The device whose attributes are given as subscription points must also be running.
- Ending a scheduling block requires the presence of the FSPs that were assigned to the sub-array and any device whose attributes were given as subscription points.

Starting and ending a scan do not explicitly require the presence of the assigned VCCs and FSPs. Any device that is unreachable will simply be skipped.

7.2.6 Interfaces

Mid.CBF Subarray (and, in particular, `CbfSubarray`) implements a set of properties, attributes, and commands as an interface feature.

7.2.6.1 Properties

Mid.CBF Subarray implements the set of properties listed in TABLE 7-4.

Table 7-4 Mid.CBF Subarray Properties

Name	Type	Description
SubID	`uint16`	The ID of the Mid.CBF Subarray.
CbfMasterAddress	`str`	The FQDN of Mid.CBF Master.

Name	Type	Description
SW1Address	`str`	The FQDN of the Mid.CBF Subarray's first search window capability.
SW2Address	`str`	The FQDN of the Mid.CBF Subarray's second search window capability.
VCC	`(str,)`	An array of the FQDNs of all the Mid.CBF VCC Capabilities (including those not affiliated with the particular Mid.CBF Subarray).
FSP	`(str,)`	An array of the FQDNs of all the Mid.CBF FSP Capabilities (including those not affiliated with the particular Mid.CBF Subarray).
FspSubarray	`(str,)`	An array of the FQDNs of the Mid.CBF FSP Subarray Capabilities that belong to the Mid.CBF Subarray.

7.2.6.2 Attributes

Mid.CBF Subarray implements the set of attributes listed in TABLE 7-5.

Table 7-5 Mid.CBF Subarray Attributes

Name	Type	Access	Description
scanID	`uint`	R/O	The ID of the scan currently being executed (0 if no scan is active).
frequencyBand	`DevEnum`	R/O	The frequency band being observed by the current scan. The enumeration is as follows: 0: "1" 1: "2" 2: "3" 3: "4" 4: "5a" 5: "5b"
receptors	`(uint16,)`	R/W	An array of the receptor IDs currently assigned to the Mid.CBF Subarray.
vccState	`(DevState,)`	R/O	An array of the states of each VCC currently affiliated with the Mid.CBF Subarray.
vccHealthState	`(uint16,)`	R/O	An array of the health states of each VCC currently affiliated with the Mid.CBF Subarray.

Name	Type	Access	Description
fspState	(DevState,)	R/O	An array of the states of each FSP currently affiliated with the Mid.CBF Subarray.
fspHealthState	(uint16,)	R/O	An array of the health states of each FSP currently affiliated with the Mid.CBF Subarray.
fspList	(uint16,)	R/O	List of FSPs used by subarray
latestScanConfig	(uint16,)	R/O	Saves the last scan configuration. So that we can refer to it even if the device is deconfigured
configID	str	R/O	Configuration ID. Comes with each configureScan JSON, in which it is called "id".

7.2.6.3 Commands

Mid.CBF Subarray implements the set of commands listed in TABLE 7-6.

Note:

- CBF Subarray uses Base Class 0.6.0 new scheme. Most of the output is a success message. In the old scheme, in most cases there will be no output.
- With the new scheme, a certain command is allowed or not is determined by the base class according to the current state. So there is no "is_xxx_allowed" function implemented.

Table 7-6 Mid.CBF Subarray Commands

Name	Input Type	Output Type	Description
On	None	DevVarLongStringArray(message)	Enable the Mid.CBF Subarray (which sends it to ON state) and turn on its two search window capabilities.
Off	None	DevVarLongStringArray(message)	Disable the Mid.CBF Subarray (which sends it to DISABLE state) and turn off its two search window capabilities.
AddReceptors	(uint16,)	DevVarLongStringArray(message)	Add the specified receptors to the Mid.CBF Subarray. Allowed only when the Mid.CBF Subarray is enabled and its observing state is EMPTY / IDLE.

Name	Input Type	Output Type	Description
RemoveReceptors	`(uint16,)`	`DevVarLongStringArray(message)`	Remove the specified receptors from the Mid.CBF Subarray. Allowed only when the Mid.CBF Subarray is enabled and its observing state is IDLE.
RemoveAllReceptors	`None`	`DevVarLongStringArray(message)`	Remove all the receptors currently assigned to the Mid.CBF Subarray. The Observing state will consequently turn to EMPTY Allowed only when the Mid.CBF Subarray is enabled and its observing state is IDLE.
ConfigureScan	`str`	`DevVarLongStringArray(message)`	Configure a scan. The input is a serialized JSON object with the scan parameters. An example is given in SECTION 15.1. Allowed only when the state of the Mid.CBF Subarray is ON and its observing state is IDLE or READY.
ConfigureSearchWindow	`str`	`None`	Configure a search window. The input is a serialized JSON object with the search window parameters. An example is given in SECTION 15.1. Allowed only when the state of the Mid.CBF Subarray is ON and its observing state is CONFIGURING. Note: This command is internal to the Mid.CBF MCS.
Scan	`int`	`DevVarLongStringArray(message)`	Start the scan with a specific scan ID. Allowed only when the state of the Mid.CBF Subarray is ON and its observing state is READY.
EndScan	`None`	`DevVarLongStringArray(message)`	End the scan. Allowed only when the state of the Mid.CBF Subarray is ON and its observing state is SCANNING.
Abort	`None`	`DevVarLongStringArray(message)`	Abort From IDLE, CONFIGURING, READY, or SCANNING ObsState, and enter state ABORTED.
GoToIdle	`None`	`DevVarLongStringArray(message)`	Deconfigure the scan configuration. Set ObsState from READY to IDLE.

Name	Input Type	Output Type	Description
ObsReset	None	DevVarLongStringArray(message)	Deconfigure the scan configuration. From ObsState ABORTED or FAULT, reset ObsState to IDLE.
Restart	None	DevVarLongStringArray(message)	Deconfigure and release all the receptors. From ObsState ABORTED or FAULT, reset ObsState to EMPTY.

7.2.7 Resources

Mid.CBF Subarray requires no particular resources to operate.

7.2.8 References

AD1 and AD2 provide more details on the design and interfaces of Mid.CBF Subarray.

7.2.9 Data

All important and relevant data of Mid.CBF Subarray has been exposed as attributes (SECTION 7.2.6.2).

7.3 Mid.CBF VCC Capability

Mid.CBF VCC Capability registers in the TANGO database with the following FQDN:

`mid_csp_cbf/vcc/xxx`

where `xxx` refers to instances `001` through `197` (though only `001` through `004` are implemented as part of the MVP).

7.3.1 Type

Mid.CBF VCC Capability is a TANGO Device Server implemented predominantly by the device class `Vcc` (though subordinate devices exist, see SECTION 7.3.4).

7.3.2 Development Type

Mid.CBF VCC Capability is newly-developed, open-source software.

7.3.3 Function and Purpose

Mid.CBF VCC Capability provides a high-level interface for monitor and control of a VCC on a TALON-DX board. It implements commands and attributes needed for scan configuration and execution, and in particular allows the configuration of the observed frequency band, search windows, and delay model updates.

7.3.4 Subordinates

Mid.CBF VCC Capability has six subordinate devices. Four of these devices instantiate the `VccBand1And2`, `VccBand3`, `VccBand4`, and `VccBand5` device classes that specify the operative frequency band of a scan. They register in the TANGO database with the following FQDNs:

```
mid_csp_cbf/vcc_band12/xxx
mid_csp_cbf/vcc_band3/xxx
mid_csp_cbf/vcc_band4/xxx
mid_csp_cbf/vcc_band5/xxx
```

where `xxx` refers to instances `001` through `197` (though only `001` through `004` are implemented as part of the MVP).

The remaining two subordinate devices both instantiate the `VccSearchWindow` device class, which implement attributes that specify parameters of the two search windows on a VCC. They register in the TANGO database with the following FQDNS:

```
mid_csp_cbf/vcc_sw1/xxx
mid_csp_cbf/vcc_sw2/xxx
```

where `xxx` refers to instances `001` through `197` (though only `001` through `004` are implemented as part of the MVP).

7.3.5 Dependencies

Mid.CBF VCC Capability has no dependencies on startup. It may be initialized before or after the FSP Capabilities, Mid.CBF Subarrays, or Mid.CBF Master. Additionally, no dependencies on other devices exist for command execution.

7.3.6 Interfaces

Mid.CBF VCC Capability (and, in particular, `Vcc`) implements a set of properties, attributes, and commands as an interface feature.

7.3.6.1 Properties

Mid.CBF VCC Capability implements the set of properties listed in TABLE 7-7.

Table 7-7 Mid.CBF VCC Capability Properties

Name	Type	Description
VccID	`uint16`	The ID of the Mid.CBF VCC Capability.
Band1And2Address	`str`	The FQDN of the VCC's capability for observing in frequency bands "1" and "2".
Band3Address	`str`	The FQDN of the VCC's capability for observing in frequency band "3".
Band4Address	`str`	The FQDN of the VCC's capability for observing in frequency band "4".
Band5Address	`str`	The FQDN of the VCC's capability for observing in frequency bands "5a" and "5b".
SW1Address	`str`	The FQDN of the VCC's first search window capability.
SW2Address	`str`	The FQDN of the VCC's second search window capability.

7.3.6.2 Attributes

Mid.CBF VCC Capability implements the set of attributes listed in TABLE 7-8.

Table 7-8 Mid.CBF VCC Capability Attributes

Name	Type	Access	Description
receptorID	`uint16`	R/O	The ID of the VCC's correponding receptor.
subarrayMembership	`uint16`	R/W	The sub-array affiliation of the VCC (0 if no affiliation).
frequencyBand	`DevEnum`	R/O	The frequency band being observed by the current scan. The enumeration is as follows: 0: "1" 1: "2" 2: "3" 3: "4" 4: "5a" 5: "5b"
band5Tuning	`(float,)`	R/W	The stream tuning for frequency bands "5a" and "5b", in GHz. The first element corresponds to the first stream, and the second element corresponds to the second stream.

frequencyBandOffsetStream1	int	R/W	The frequency offset of the first stream, in Hz.
frequencyBandOffsetStream2	int	R/W	The frequency offset of the second stream, in Hz.
delayModel	((float,),)	R/O	The current active delay model for each of 26 frequency slices. Each delay model consists of 6 coefficients representing a fifth-order polynomial.
configID	str	R/W	Configuration ID. It is set when subarray issues command configureScan. The input JSON propogates information to VCC, in which it is called “id”.
scanID	int	R/W	ID of a scan execution, propagated from subarray when transition to SCANNING is performed.

7.3.6.3 Commands

Mid.CBF VCC Capability implements the set of commands listed in TABLE 7-9.

Table 7-9 Mid.CBF VCC Capability Commands

Name	Input Type	Output Type	Description
On	None	None	Turn on the VCC, including its frequency band and search window capabilities. Allowed only when the state of the VCC is OFF and its observing state is IDLE.
Off	None	None	Turn off the VCC, including its frequency band and search window capabilities. Allowed only when the state of the VCC is ON and its observing state is IDLE.
SetFrequencyBand	str	None	Set the frequency band of the VCC. The corresponding frequency band capability enters the ON state, and all others enter the DISABLE state. Allowed only when the state of the VCC is ON and its observing state is CONFIGURING.
SetObservingState	uint16	None	Set the observing state of the VCC to CONFIGURING (1) or READY (2). Allowed only when the state of the VCC is ON and its observing state is IDLE, CONFIGURING, or READY.

Name	Input Type	Output Type	Description
			Note: This command is internal to the Mid.CBF MCS.
UpdateDelayModel	str	None	Update the VCC's delay model. The input is a serialized JSON object with the delay model coefficients. An example is given in SECTION 15.2. Allowed only when the state of the VCC is ON and its observing state is READY or SCANNING. Note: This command is internal to the Mid.CBF MCS.
ValidateSearchWindow	str	None	Validate a search window configuration. The input is a serialized JSON object with the search window parameters. An example is given in SECTION 15.1. Allowed in all states and observing states. Note: This command is internal to the Mid.CBF MCS.
ConfigureSearchWindow	str	None	Configure a search window. The input is a serialized JSON object with the search window parameters. An example is given in SECTION 15.1. Allowed only when the state of the VCC is ON and its observing state is CONFIGURING. Note: This command is internal to the Mid.CBF MCS.
Scan	Uint16	None	Start the scan.(controlled by the subarray) Allowed only when the state of the VCC is ON and its observing state is READY.
EndScan	None	None	End the scan. Allowed only when the state of the VCC is ON and its observing state is SCANNING.
GoToIdle	None	None	Transition to an IDLE observing state. Allowed only when the state of the VCC is ON and its observing state is IDLE or READY.

7.3.7 Resources

Mid.CBF VCC Capability requires no particular resources to operate.

7.3.8 References

AD1 and AD2 provide more details on the design and interfaces of Mid.CBF VCC Capability.

7.3.9 Data

All important and relevant data of Mid.CBF VCC Capability has been exposed as attributes (SECTION 7.3.6.2).

7.4 Mid.CBF FSP Capability

Mid.CBF FSP Capability registers in the TANGO database with the following FQDN:

```
mid_csp_cbf/fsp/xx
```

where `xx` refers to instances `01` through `27` (though only `01` through `04` are implemented as part of the MVP).

7.4.1 Type

Mid.CBF FSP Capability is a TANGO Device Server implemented predominantly by the device class `Fsp` (though subordinate devices exist, see SECTION 7.4.4).

7.4.2 Development Type

Mid.CBF FSP Capability is newly-developed, open-source software.

7.4.3 Function and Purpose

Mid.CBF FSP Capability provides a high-level interface for monitor and control of an FSP on a TALON-DX board. It implements commands and attributes needed for scan configuration and execution, and in particular allows the configuration of the function mode (only CORRELATION for the MVP) and enables signal processing capability during a scan.

7.4.4 Subordinates

Mid.CBF FSP Capability has a number of subordinate devices. Four of these devices instantiate the `FspCorr`, `FspPss`, `FspPst`, and `FspVlbi` device classes that specify the FSP's operative function mode. They register in the TANGO database with the following FQDNs:

```
mid_csp_cbf/fsp_corr/xx
mid_csp_cbf/fsp_pss/xx
```

```
mid_csp_cbf/fsp_pst/xx
mid_csp_cbf/fsp_vlbi/xx
```

where `xx` refers to instances `01` through `27` (though only `01` through `04` are implemented as part of the MVP).
The remaining subordinate devices instantiate the `FspSubarray` device class, which is detailed in SECTION 7.5.

7.4.5 Dependencies

The dependencies of Mid.CBF FSP Subarray Capability are detailed separately in SECTION 7.5.5.
Mid.CBF FSP Capability has no dependencies on startup. It may be initialized before or after the VCC Capabilities, Mid.CBF Subarrays, or Mid.CBF Master. Additionally, no dependencies on other devices exist for command execution.

7.4.6 Interfaces

Mid.CBF FSP Capability (and, in particular, `Fsp`) implements a set of properties, attributes, and commands as an interface feature.

7.4.6.1 Properties

Mid.CBF FSP Capability implements the set of properties listed in TABLE 7-10.

Table 7-10 Mid.CBF FSP Capability Properties

Name	Type	Description
FspID	`uint16`	The ID of the Mid.CBF FSP Capability.
CorrelationAddress	`str`	The FQDN of the FSP's capability for correlation.
PSSAddress	`str`	The FQDN of the FSP's capability for PSS.
PSTAddress	`str`	The FQDN of the FSP's capability for PST.
VLBIAddress	`str`	The FQDN of the FSP's capability for VLBI.
FspSubarray	`(str,)`	An array of the FQDNs of the Mid.CBF FSP Subarray Capabilities that belong to the FSP.

7.4.6.2 Attributes

Mid.CBF FSP Capability implements the set of attributes listed in TABLE 7-11.

Table 7-11 Mid.CBF FSP Capability Attributes

Name	Type	Access	Description
functionMode	`DevEnum`	R/O	The current function mode of the FSP. The enumeration is as follows: 0: “IDLE” 1: “CORRELATION” 2: “PSS” 3: “PST” 4: “VLBI”
subarrayMembership	`(uint16,)`	R/O	An array of the sub-array affiliations of the FSP (empty if no affiliation).

7.4.6.3 Commands

Mid.CBF FSP Capability implements the set of commands listed in TABLE 7-12.

Table 7-12 Mid.CBF FSP Capability Commands

Name	Input Type	Output Type	Description
On	`None`	`None`	Turn on the FSP, including its function mode and FSP Subarray capabilities. Allowed only when the state of the FSP is OFF.
Off	`None`	`None`	Turn off the FSP, including its function mode and FSP Subarray capabilities. Allowed only when the state of the FSP is ON.
SetFunctionMode	`str`	`None`	Set the function mode of the FSP. The corresponding function mode capability enters the ON state, and all others enter the DISABLE state. Allowed only when the state of the FSP is ON.
AddSubarrayMembership	`uint16`	`None`	Affiliate the FSP with a sub-array. Allowed only when the state of the FSP is ON. Note: This command is internal to the Mid.CBF MCS.
RemoveSubarrayMembership	`uint16`	`None`	Disaffiliate the FSP from a sub-array. Allowed only when the state of the FSP is ON.

			Note: This command is internal to the Mid.CBF MCS.
getConfigID	None	DevString	Returns a JSON object as a string. Includes all the fspCorrSubarray and their configID attribute.

7.4.7 Resources

Mid.CBF FSP Capability requires no particular resources to operate.

7.4.8 References

AD1 and AD2 provide more details on the design and interfaces of Mid.CBF FSP Capability.

7.4.9 Data

All important and relevant data of Mid.CBF FSP Capability has been exposed as attributes (SECTION 7.4.6.2).

7.5 Mid.CBF FSP Subarray Capability

Mid.CBF FSP Subarray Capability registers in the TANGO database with the following FQDN:

```
mid_csp_cbf/fspCorrSubarray/xx_yy
mid_csp_cbf/fspPssSubarray/xx_yy
mid_csp_cbf/fspPstSubarray/xx_yy
mid_csp_cbf/fspVlbiSubarray/xx_yy
```

where `xx` refers to FSP instances `01` through `27` and `yy` refers to sub-array instances `01` through `16` (though only FSPs `01` through `04` and sub-array `01` are implemented as part of the MVP).

7.5.1 Type

Mid.CBF FSP Subarray Capability is a TANGO Device Server implemented by the device class
`FspCorrSubarray`.
`FspPssSubarray`.
`FspVlbiSubarray`.
`FspPstSubarray`.

7.5.2 Development Type

Mid.CBF FSP Subarray Capability is newly-developed, open-source software.

7.5.3 Function and Purpose

Mid.CBF FSP Subarray Capability implements commands and attributes needed for scan configuration and execution on an FSP.

7.5.4 Subordinates

Mid.CBF FSP Subarray Capability has no immediate subordinates.

7.5.5 Dependencies

Mid.CBF FSP Subarray Capability has no dependencies on startup. It may be initialized before or after the VCC Capabilities, Mid.CBF Subarrays, or Mid.CBF Master. However, the following dependencies exist for commands:

- Adding receptors requires the presence of Mid.CBF Master and the receptors' corresponding VCCs that are being added or removed.

7.5.6 Interfaces

Mid.CBF FSP Subarray Capability implements a set of properties, attributes, and commands as an interface feature.

7.5.6.1 Properties

Mid.CBF FspCorrSubarray Capability implements the set of properties listed in TABLE 7-13.

Table 7-13 Mid.CBF FspCorrSubarray Capability Properties

Name	Type	Description
SubID	`uint16`	The Subarray ID of the Mid.CBF FSP Subarray Capability.
FspID	`uint16`	The FSP ID of the Mid.CBF FSP Subarray Capability.
CbfMasterAddress	`str`	The FQDN of Mid.CBF Master.
CbfSubarrayAddress	`str`	The FQDN of the Mid.CBF Subarray that the FSP Subarray belongs to.
VCC	`(str,)`	An array of the FQDNs of all the Mid.CBF VCC Capabilities.

7.5.6.2 Attributes

Mid.CBF FspCorrSubarray Capability implements the set of attributes listed in TABLE 7-14.

Table 7-14 Mid.CBF FspCorrSubarray Capability Attributes

Name	Type	Access	Description
receptors	`(uint16,)`	R/W	An array of the receptor IDs currently assigned to the FSP Subarray.
frequencyBand	`DevEnum`	R/O	The frequency band being observed by the current scan. The enumeration is as follows: 0: “1” 1: “2” 2: “3” 3: “4” 4: “5a” 5: “5b”
band5Tuning	`(float,)`	R/O	The stream tuning for frequency bands “5a” and “5b”, in GHz. The first element corresponds to the first stream, and the second element corresponds to the second stream.
frequencyBandOffsetStream1	`int`	R/O	The frequency offset of the first stream, in Hz.
frequencyBandOffsetStream2	`int`	R/O	The frequency offset of the second stream, in Hz.
frequencySliceID	`uint16`	R/O	The frequency slice being correlated.
corrBandwidth	`uint16`	R/O	The bandwidth to be correlated is the full bandwidth of the frequency slice divided by 2 raised to the power of this attribute’s value. For example, if the value of this attribute is 4, only a quarter of the full frequency slice is correlated.
zoomWindowTuning	`uint`	R/O	The center frequency of the spectral zoom window, in kHz, if not the full frequency slice is being correlated.
integrationTime	`uint16`	R/O	The integration time, in milliseconds.
channelAveragingMap	`((uint16,),)`	R/O	The channel averaging map currently being used. Each element in the array is a tuple, where the first element is

Name	Type	Access	Description
			the ID of the first channel in a channel group for which the channel averaging factor given in the second element is applicable.
visDestinationAddress	`str`	R/W	Stores Destination addresses for visibilities, given as a JSON object
fspChannelOffset	`DevLong`	R/W	Fsp channel offset. To specify the starting fsp channel. Typically a multiple of 14480.
outputLinkMap	`((‘DevUlong64’))`	R/O	2*40 array for outputLinkMap
scanID	`DevLong64`	R/W	scanID, propagated from subarray when transition to SCANNING is performed
configID	`str`	R/W	scanID, propagated from subarray when transition to READY is performed

7.5.6.3 Commands

Mid.CBF FspCorrSubarray Capability implements the set of commands listed in TABLE 7-15.

Table 7-15 Mid.CBF FspCorrSubarray Capability Commands

Name	Input Type	Output Type	Description
On	`None`	`None`	Turn on the FSP Subarray. Allowed only when the state of the FSP Subarray is OFF and its observing state is IDLE.
Off	`None`	`None`	Turn of the FSP Subarray. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE.
AddReceptors	`(uint16,)`	`None`	Add the specified receptors to the FSP Subarray. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE, CONFIGURING, or READY. Note: This command is internal to the Mid.CBF MCS.
RemoveReceptors	`(uint16,)`	`None`	Remove the specified receptors from the FSP Subarray.

Name	Input Type	Output Type	Description
			Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE, CONFIGURING, or READY. Note: This command is internal to the Mid.CBF MCS.
RemoveAllReceptors	None	None	Remove all the receptors currently assigned to the FSP Subarray. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE, CONFIGURING, or READY. Note: This command is internal to the Mid.CBF MCS.
AddChannels	str	None	Add channel frequency information to an FSP Subarray. The input is a serialized JSON object with the required information. An example is given in SECTION 0. Allowed only when the state of the FSP Subarray is ON and its observing state is CONFIGURING. Note: This command is internal to the Mid.CBF MCS.
AddChannelAddresses	str	None	Add channel address information to an FSP Subarray. The input is a serialized JSON object with the required information. An example is given in SECTION 0. Allowed only when the state of the FSP Subarray is ON and its observing state is CONFIGURING. Note: This command is internal to the Mid.CBF MCS.
ConfigureScan	str	None	Configure a scan. The input is a serialized JSON object with the scan parameters. An example is given in SECTION 15.1. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE or READY. Note: This command is internal to the Mid.CBF MCS.
Scan	unit16	None	Start the scan. (signal is sent by the subarray) Allowed only when the state of the FSP Subarray is ON and its observing state is READY.

Name	Input Type	Output Type	Description
EndScan	None	None	End the scan. Allowed only when the state of the FSP Subarray is ON and its observing state is SCANNING.
GoToIdle	None	None	Transition to an IDLE observing state. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE or READY.
GetLinkAndAddress	None	Str(JSON format)	Get destination addresses in the form of JSON for this fspCorrSubarray. These information is given to fspCorrSubarray when a subarray issues configreScan Command. The output will be in the format of: {“outputlink”: 0, “outputHost”: “”, outputMac “”, outputPort: 0}

7.5.6.4 Properties

Mid.CBF FspPssSubarray Capability implements the set of properties listed in TABLE 7-13.

Table 7-16 Mid.CBF FspPssSubarray Capability Properties

Name	Type	Description
SubID	uint16	The Subarray ID of the Mid.CBF FSP Subarray Capability.
FspID	uint16	The FSP ID of the Mid.CBF FSP Subarray Capability.
CbfMasterAddress	str	The FQDN of Mid.CBF Master.
CbfSubarrayAddress	str	The FQDN of the Mid.CBF Subarray that the FSP Subarray belongs to.
VCC	(str,)	An array of the FQDNs of all the Mid.CBF VCC Capabilities.

7.5.6.5 Attributes

Mid.CBF FspPssSubarray Capability implements the set of attributes listed in TABLE 7-14.

Table 7-17 Mid.CBF FspPssSubarray Capability Attributes

Name	Type	Access	Description
receptors	(uint16,)	R/W	An array of the receptor IDs currently assigned to the FSP Subarray.
searchBeams	(str,)	R/O	List of searchBeams assigned to FspSubarray in JSON format
searchWindowID	(uint16,)	R/O	Identifier of the Search Window to be used as input for beamforming on this FSP.
searchBeamID	(uint16,)	R/O	List of SearchBeam Id's being used by the FspPssSubarray
outputEnable	int	R/O	Enable/disable transmission of output products.
frequencySliceID	uint16	R/O	The frequency offset of the second stream, in Hz.

7.5.6.6 Commands

Mid.CBF FspPssSubarray Capability implements the set of commands listed in TABLE 7-15.

Table 7-18 Mid.CBF FspPssSubarray Capability Commands

Name	Input Type	Output Type	Description
On	None	None	Turn on the FSP Subarray. Allowed only when the state of the FSP Subarray is OFF and its observing state is IDLE.
Off	None	None	Turn of the FSP Subarray. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE.
AddReceptors	(uint16,)	None	Add the specified receptors to the FSP Subarray. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE, CONFIGURING, or READY. Note: This command is internal to the Mid.CBF MCS.
RemoveReceptors	(uint16,)	None	Remove the specified receptors from the FSP Subarray. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE, CONFIGURING, or READY.

Name	Input Type	Output Type	Description
			Note: This command is internal to the Mid.CBF MCS.
RemoveAllReceptors	None	None	Remove all the receptors currently assigned to the FSP Subarray. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE, CONFIGURING, or READY. Note: This command is internal to the Mid.CBF MCS.
ConfigureScan	str	None	Configure a scan. The input is a serialized JSON object with the scan parameters. An example is given in SECTION 15.1. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE or READY. Note: This command is internal to the Mid.CBF MCS.
Scan	None	None	Start the scan. Allowed only when the state of the FSP Subarray is ON and its observing state is READY.
EndScan	None	None	End the scan. Allowed only when the state of the FSP Subarray is ON and its observing state is SCANNING.
GoToIdle	None	None	Transition to an IDLE observing state. Allowed only when the state of the FSP Subarray is ON and its observing state is IDLE or READY.

7.5.7 Resources

Mid.CBF FSP Subarray Capability requires no particular resources to operate.

7.5.8 References

AD1 and AD2 provide more details on the design and interfaces of Mid.CBF FSP Subarray Capability.

7.5.9 Data

Most important and relevant data of Mid.CBF FSP Subarray Capability has been exposed as attributes (SECTION 7.5.6.2).
Internally, the attribute `visDestinationAddress` is stored as a table where, for each fine channel configured to be sent to SDP, the channel's ID, bandwidth, center frequency, assigned Mid.CBF output link ID, assigned SDP host IP, and assigned SDP host port are listed.

7.6 Internal Interfaces

Refer to SECTION 6.3 for details on software behavior, including descriptions of message flow between the various components of the Mid.CBF MCS.

7.7 Requirements to Design Components Traceability

Traceability matrices have not yet been produced.

8 Control and Monitor Parameters, Indicators and Messages

This section is largely inapplicable to the Mid.CBF MCS; the purpose of the MCS itself is to monitor and control the TALON-DX boards that implement signal processing functionality. The operational health and state reported by each TANGO Device reports the health and state of the devices themselves, as well as any subordinate devices and components. This has been detailed in SECTION 6.3 and CHAPTER 7.

9 WebJive User Interfaces

The Mid.CBF MCS provides a WebJive GUI (FIGURE 9-1), which can be accessed through `localhost:22484/testdb/` once the Docker containers are running (refer to **SECTION 10**). The devices can be explored, their states and attributes viewed (FIGURE 9-2), and commands sent (FIGURE 9-3).
Note the WebJive interface is out of date as of August 2020.
Developers usually use Jive to visualize the TANGO objects (described in the "user guide" chapter).

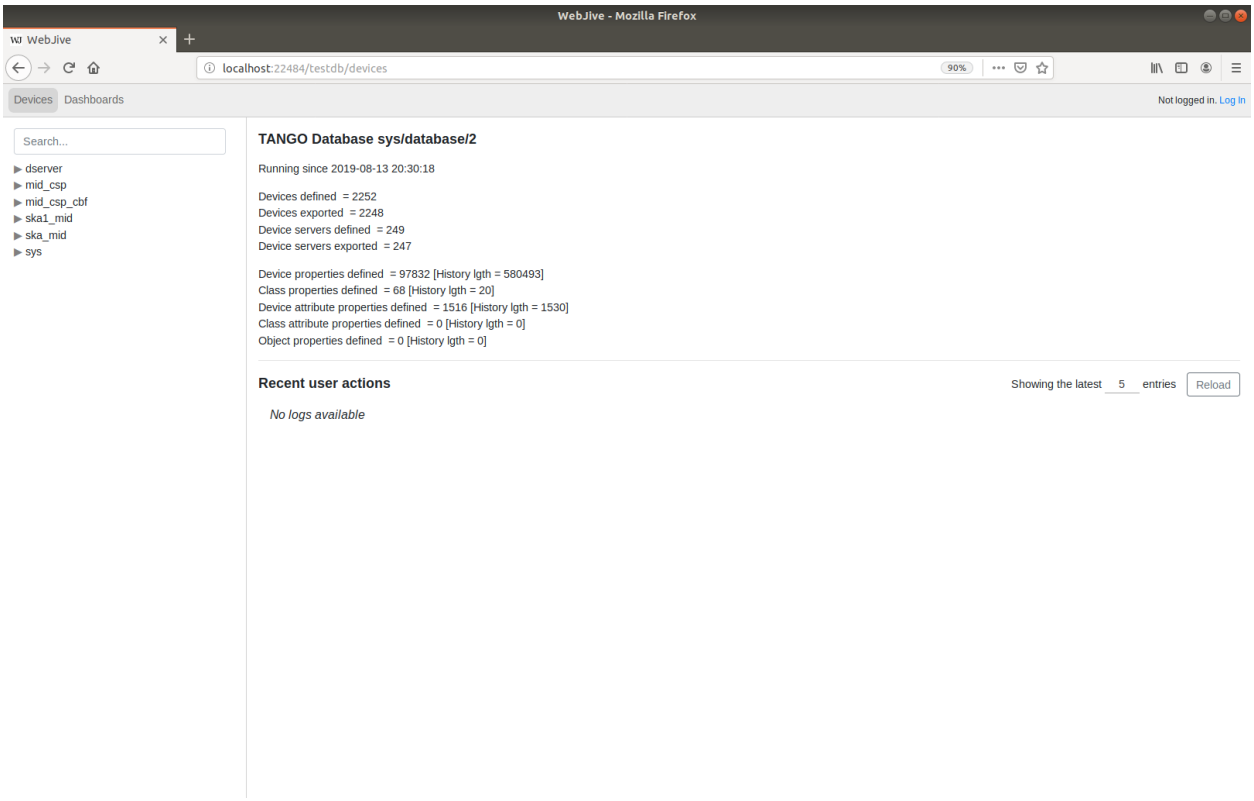

Figure 9-1 WebJive GUI landing

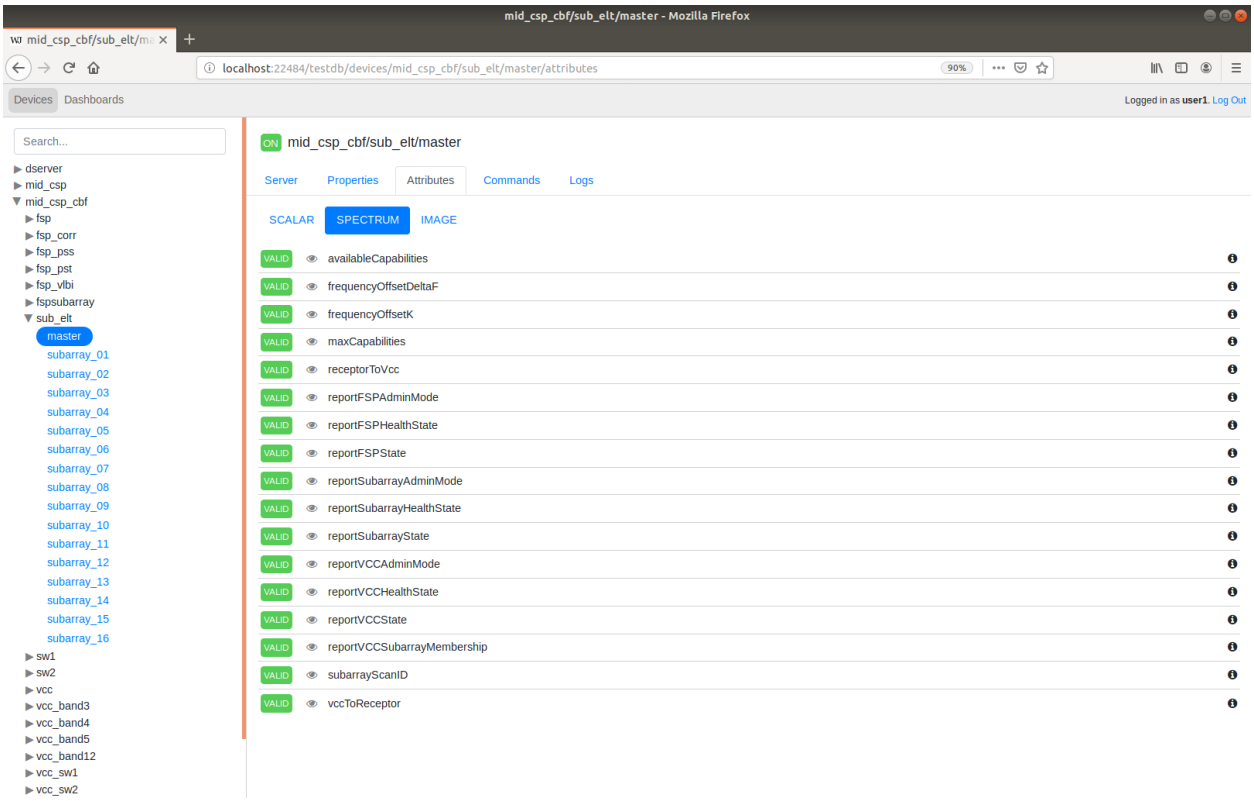

Figure 9-2 WebJive GUI viewing attributes

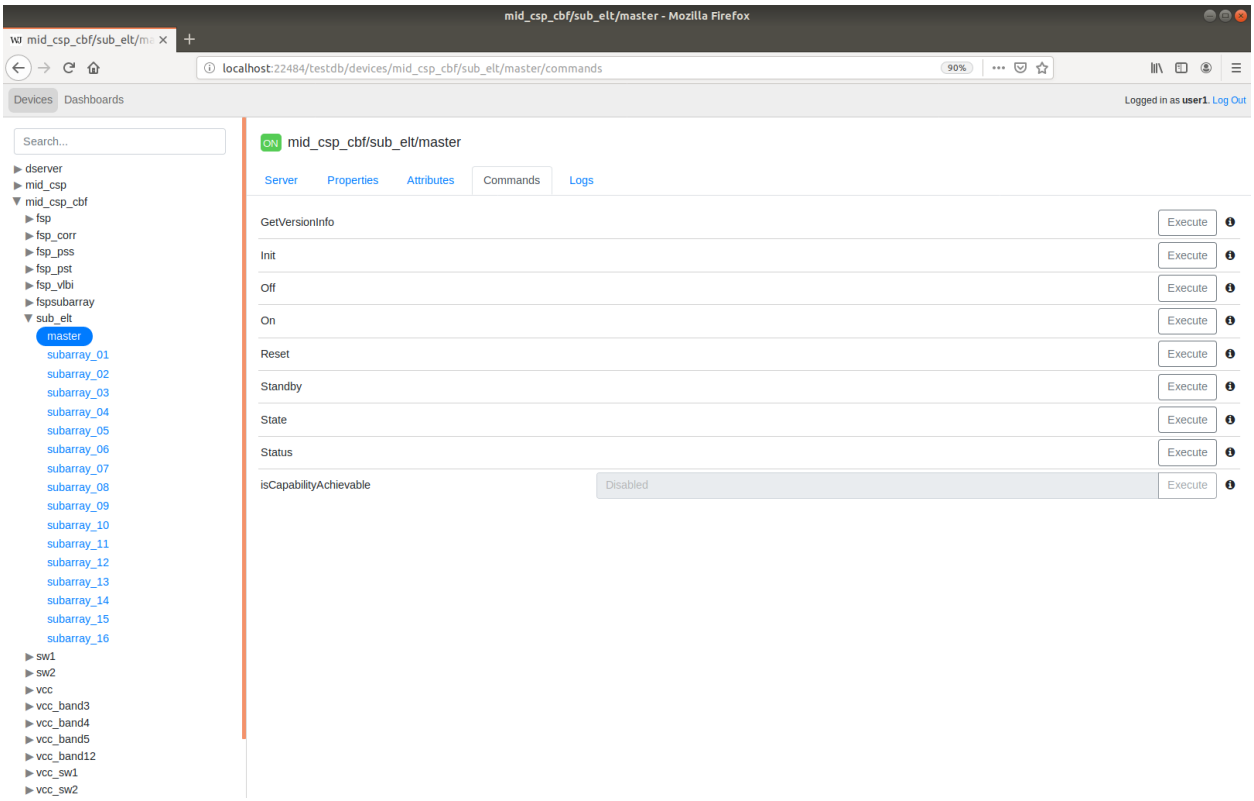

Figure 9-3 WebJive GUI sending commands

Dashboards can be created to display custom information (FIGURE 9-4), though this feature currently has limited functionality. Additionally, since these are stored and persist locally, none are provided out-of-the-box.

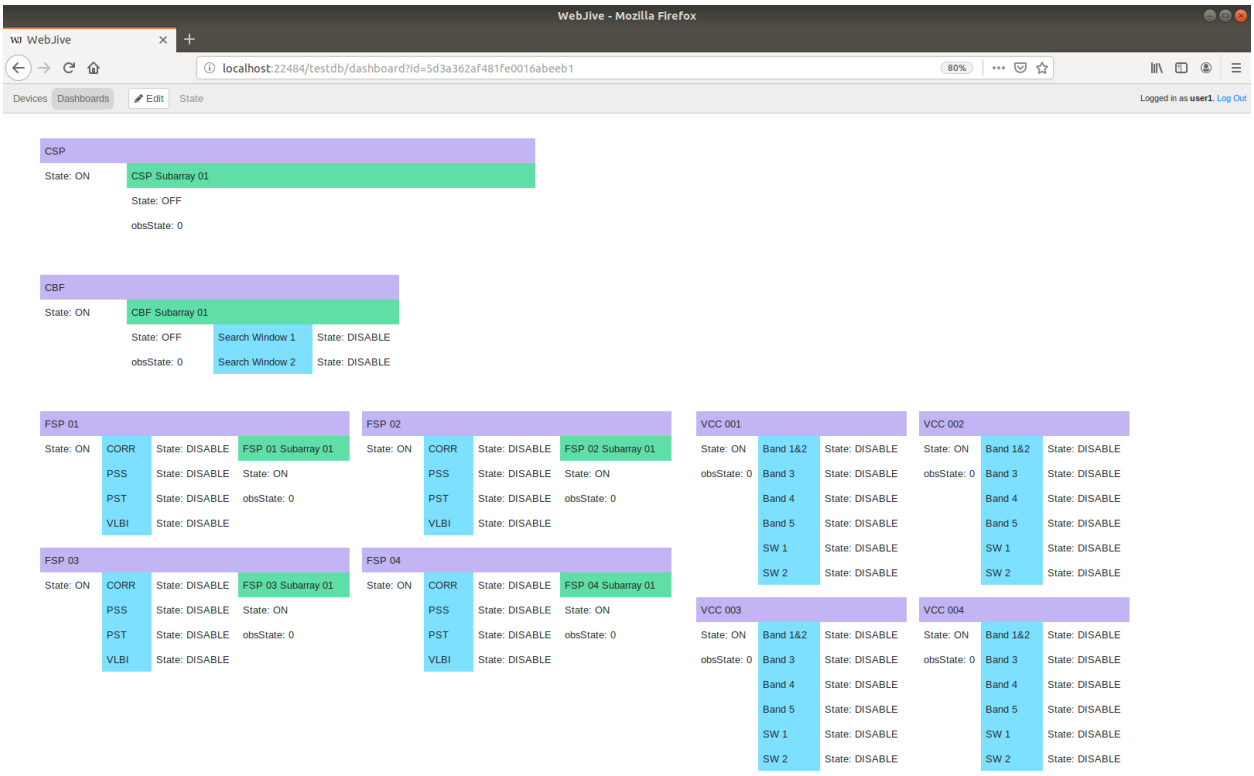

Figure 9-4 WebJive GUI custom dashboard

10 User Guide and Development Environment

This chapter details the set-up of a development and run-time environment for the Mid.CBF MCS. It is a developers' guide to recreate the same working environment and provides developers some basic knowledge.

There are several documentation that you need to read to get a feel of what mid-CBF-MCS is.

1. Look at the Mid-CBF-MCS power point for the Aug.2020 HAA tech talk, that should provide a big picture about the project and software
2. TANGO documentation. TANGO is the main framework we use in this software. The documentation is very long, read at least chapter 4 device server, that is the most important chapter
3. This documentation(Mid-CBF-MCS MVP): chapter 6-7 spefically. This is the description of the software, very useful for developer who is new and will be working on the project.
4. ICD & DDD: This is the high level design specification of CSP, only part of it apply to Mid-CBF-MCS. Make sure to pay more attention to those chapters.
5. Learn basic idea about docker, and docker commands. You will be working with only a few basic commands most of the times. You can find them later in this chapter.

At the same time, you should download the system and run it yourself. The following is a guide on how to download, start, run, develop and publish the Mid-CBF-MCS software.

10.1 Setting Up Ubuntu Subsystem

To compile this project and begin development you need to set up your IDE (Integrated development environment). The first step in this setup is downloading and running Ubuntu 18.04 on a virtual machine. An Ubuntu image is essentially a snapshot of an operating system that you will be running on a virtual box, which is a program used to run different OS on a windows computer (similar to bootcamp for MacOS). The following ubuntu link downloads the .vdi image with a preset password "osboxes.org".

Virtualbox Download: VirtualBox

Ubuntu Image Download: Ubuntu Image

Steps:

1. Install virtual box

2. Open up file downloaded from sourceforge for the ubuntu image with 7-zip and extract the "Ubuntu 18.04.2 (64bit).vdi" file into a known directory
3. Open up the virtual box software and click "new" and run through the setup process, on the Hard Disk option screen choose "use and existing virtual hard disk file" and then choose the VDI file that you extracted in step two.
4. Run the OS in virtualbox and login to the ubuntu OS. The login screen should show the account "osboxes.org" it will ask for a password, this is a default account the virtual machine creates for you and the password is "osboxes.org" (you can change the name and password in account settings once you are logged in")

Note: You can also choose to download .iso file from the offcial Ubuntu site, that way you can set your own preferences and password from the beginning.

10.1.1 Improving Virtual Machine Performance

Most of the development will be done in this ubuntu subsystem so allowing access to more system resources, escpecially for applications that require more ram and cpu usage

1. Open VirtualBox and open up settings on the VM you want to improve performance on:
2. Open the system tab:
    i. Motherboard Tab: Increase base memory to the end of the green line
    ii. Processor Tab:Increase processors to the end of the green line
3. Screen Tab: Increase base memory to the end of the green line
    i. Screen Tab: Increase base memory to the end of the green line
4. Open the General Tab:
    i. Advanced Tab: Shared Clipboard: Bidirectional (you can now copy and paste things between windows and ubuntu)

10.1.2 Sharing Files Between Windows and VirtualBox Ubuntu

Throughout the development proccess there will be instances where you want to bring files between the windows machine and the ubuntu subsystem setting up this folder that allows for file transfer between systems is very important and can be done following the steps listed below.

1. Open VirtualBox and open up settings on the VM you want a shared folder on
2. Go to shared folders section and click add a new shared folder

3. Select a path for the shared folder on your windows machine and use folder name “shared”
4. Uncheck Read-only and Auto-mount, and check Make Permanent
5. Start up your VM and login, on the top menu bar click Devices->Insert Guest Addition CD image.
6. Use the following command to mount the CD, install dependencies and run the installation script and then reboot the VM:

```
$ sudo mount /dev/cdrom /media/cdrom
$ sudo apt-get update
$ sudo apt-get install build-essential linux-headers-`uname -r`
$  sudo /media/cdrom/./VBoxLinuxAdditions.run
$ sudo shutdown -r now
```

7. Open up console and create a shared directory and then mount the shared folder from your host into your ~/shared directory

```
$ mkdir ~/shared
$ sudo mount -t vboxsf shared ~/shared
```

This directory is only temporary and will disappear after your ned reboot, to make this file permanent follow these next steps:

8. Edit the fstab file in the etc directory and then add the following line to fstab run the command separated by tabs and then press Ctrl+O to save

```
$  sudo nano /etc/fstab
$ shared /home/<username>/shared vboxsf  defaults  0     0
Ctrl + O
```

9. Edit the modules file in the etc directory and add Vboxsf line to modules and then press Ctrl+O to save and then reboot the VMusing command:

```
$  sudo nano /etc/modules
$ Vboxsf
Ctrl + O
$ sudo shutdown –r now
```

10. Go to your home directory and check to see if the file is highlighted in green if it is then you have successfully made the folder permanent.

```
$ cd ~
$ ls
```

10.2 Software Enviroment

10.2.1 Setting up Tango Enviroment

Setting up the tango enviroment is the next step in this starting guide. Most of the enviroment set up is automated and is installed and configured using the ansible playbook script and the docker compose script. These scripts will install the programs and modules listed below:

- python version 3.7
- TANGO-controls '9.3.3'
- Visual Studio Code, PyCharm Community Edition
- ZEROMQ '4.3.2'
- OMNIORB '4.2.3'

Run the following commands in your terminal, it will clone the ansible playbook repository, install it, and then run the script and set up the enviroment, if you experience issues with the script ask questions in the #team-system-support slack channel. The following instruction can be also found at the developer's portal: https://developer.skatelescope.org/en/latest/tools/tango-devenv-setup.html

```
$ sudo apt -y install git
$ git clone https://gitlab.com/ska-telescope/ansible-playbooks
$ cd ansible-playbooks
$ sudo apt-add-repository --yes --update ppa:ansible/ansible && sudo apt -y install
ansible
$ ansible-playbook -i hosts deploy_tangoenv.yml --extra-vars
"ansible_become_pass=osboxes.org"
$ sudo reboot
```

Trouble shooting:

- If you set your own password for the virtual machine, change `"ansible_become_pass=osboxes.org" to "ansible_become_pass=<your own password"`
- `If you encounter python related problem with the ansible command, try to specify the python version with anisible script, for example:`

```
$ ansible-playbook -i hosts deploy_tangoenv.yml --extra-vars
"ansible_become_pass=osboxes.org" -e ansible_python_interpreter=/usr/bin/python2
```

Using python2 like above may solve the problem for you.

10.2.2 Setting up Mid-CBF-MCS

When the steps outlined in the previous sections are complete, the Mid.CBF MCS Gitlab repository may be cloned

```
$ git clone https://gitlab.com/ska-telescope/mid-cbf-mcs.git
$ cd mid-cbf-mcs
```

10.2.3 Setting Up LMC Base Classes

LMC Base Class is also needed if you want to use Pogo. Cloning this repository is recommended.

```
$ git clone https://gitlab.com/ska-telescope/lmc-base-classes
$ cd lmc-base-classes
```

10.3 Docker Commands

Here are a few docker commands that are useful for developers.

Each component of the Mid.CBF MCS runs in a separate Docker container, which are all created at run-time.

To test a few docker commands, first go back to Mid-CBF-MCS directory

```
$ cd mid-cbf-mcs
```

To build the image, issue command:

```
$ make build
```

To confirm the image is build:

```
$ docker image ls
```

To run the docker image:

```
$ make up
```

To see the running containers:

```
$ docker ps
```

You can view all the containers including exited onces by running

```
$ docker ps -a
```

At this stage all the devices should be running. You can follow the JIVE guide to further inspect.

It is useful to debug by looking at the logs of the containers.

```
$ docker logs -f <container>
```

To stop all the containers

```
$ make down
```

10.4 Running devices

10.4.1 Using JIVE

JIVE is a graphical user interface that visualizes the devices, servers, and executes devices' commands. Here is the procedure to start JIVE manually:

1. From the project root directory:

```
$ make build

$ make up
```

2. Run the following command

```
$ docker network inspect tangonet
```

3. Find "midcbf-databaseds", then copy the first part of its IPv4Address

4. Run the following command:

```
$ export TANGO_HOST=<the address from step 3>:100000
```

5. Run JIVE:

```
$ jive
```

Note: step 2-3 can be automated by running "python configJive.py" script in the main folder:

```
$ python configJIVE.py
```

You then have to manually copy and run the export command popped up in the terminal.

10.4.2 Running the state machine

After JIVE started, run the state machine and you will understand the system much better.
Follow the table in chapter 6 for a normal operation procedure.
To Run a device:

1. open its server, and find the device with the gear button as in figure 10-1.
2. Right click the device, and select test device
3. You will see a list of commands and attributes. Run certain commands to trigger state change.

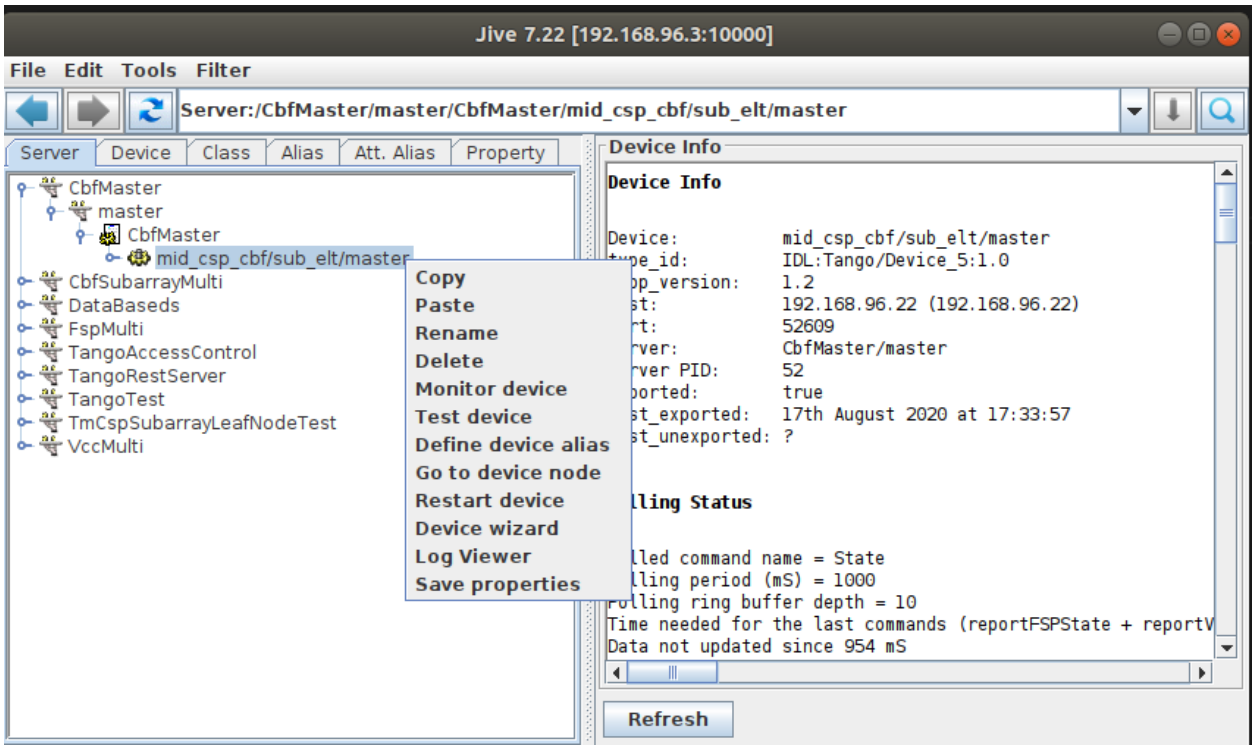

Figure 10-1 Jive GUI start device

Note: Some commands and attributes shown on Jive are there because the device inherited from base class. They are not meaningful commands for the device so don't use them. All the useful commands and attributes are described in chapter 7 of this document.

10.4.2.1 Device Relationships

To test the state machine on Jive, the most important devices are CbfSubarray and CbfMaster. In short, the Master controls the power status of all the devices, and the Subarray processes all the operations and send instructions to all other devices.

10.4.2.2 Configure scan

ConfigureScan is a special command for Mid-CBF-MCS, you need a valid JSON string to pass into Jive. Furthermore, the input in Jive needs "\" before each quotation mark. Normal JSON file wouldn't work.

To solve this probelm, there are three options:

1. Use a script to generate this specific input.
Put your JSON file in tangods/CbfSubarray/JIVEconfigscan/scanconfig.JSON.
Run :

```
$ Python generateJIVE.py
```

2. Use the sendConfig device to trigger configure scan with the subarray: put configuration JSON in tangods/CbfSubarray/sendConfig/config.JSON

3. Manually insert "\" before each quotation mark (not recommended...)

10.5 Development

10.5.1 Branching with Git

Follow the guidelines here: https://developer.skatelescope.org/en/latest/tools/git.html

Some simple instructions:
Create a new branch

```
Git checkout master

Git checkout –b <branch name>
```

Push this branch to remote. Here remote is often origin. Branch name is the new branch that you just created.

```
Git push <remote> <branch name>
```

Set up upstream tracking branch

```
Git branch –u orgin/<branch name>
```

Checkout all the branches and their tracking branch

```
Git branch -vv
```

10.5.2 Using Pogo

Pogo is used to automatically generate TANGO commands, attributes and properties. The pogo file contains a device's information, and can automatically generate corresponding python files for you. However, the pogo files for Mid-CBF-MCS does not reflect current device attributes, commands and properties.
To run pogo, go to the device directory and find the .xmi file in the pogo folder. Here are some advices and observations for Pogo:

- if running "pogo xx.xmi" gives error, then find the corresponding .xmi file in the LMC base class folder
- Pogo can generate a skeleton python file, which includes importing, device properties, and methods. In this project, not all device properties/attributes/commands are generated by pogo, so these attributes wouldn't show on the pogo GUI when you run it.
- Pogo will overwrite the original python files, that it why it is kept in another folder. You can generate a new attribute, copy the part of the python code generated, and paste into the file you want.
- In the code you will see "#protected regions", these are the regions that pogo won't overwrite
- pogo imports pytango instead of tango in the generated .py files

10.5.3 Understanding python files for Tango devices

10.5.3.1 Init_device:

- Analogous to the "__init__" function in python. You should start here to analyze the code.
- Define your own internal attributes here, to be used in TANGO attributes' methods later

Note: The new scheme for LMC base class 0.6.0 moves "init_device" to "class InitCommand". Currently only CbfSubarray is implemented with the new scheme.

10.5.3.2 Device properties

- Can be generated by pogo
- Can be manually changed. You need to know the format from pogo generated code.
- Has to be set in the config file: "/data/midcbf_dsconfig.json", or give a default value

10.5.3.3 Attribute

- Can be generated by pogo
- Access: define read or write authority

- Label: define how it appears in JIVE
- The Attributes method function is used to define what happens if the client read or write the attribute. The code can be generated by pogo. For example, if you have “receptorID” in the attribute part, and it is readable, then “read_receptorID” will be its attribute method function name.

10.5.4 Sphinx

Sphinx is used to generate documentation from comments inside the python code, to be posted on ReadTheDoc(https://developer.skatelescope.org/projects/mid-cbf-mcs/en/latest/?badge=latest)
Sphinx is already set up for this repository. However, if you are curious to learn, here are some resources:
Documentation: https://www.sphinx-doc.org/en/master/usage/quickstart.html
Pay attention to:

- The config.py file: included library, extensions, etc
- Index.rst is the main file to write
- In the .rst files, have an empty line after :xx:, otherwise gives error

10.5.5 Understanding LMC Base Class 0.6.0

CbfSubarray is updated to the new Scheme for LMC 0.6.0, all the other devices are still using the old scheme.
Helpful links to read about the steps to update to the new scheme:
https://confluence.skatelescope.org/display/SE/Updating+to+lmc-base-classes+0.6.x
https://gitlab.com/ska-telescope/lmc-base-classes

10.5.5.1 Summary of changes

If the new scheme is used, like the CbfSubarray device, then:

- In the do hook for a command class, “self” becomes “self.target”
- State change is managed by the base class. There is no need to write code to trigger state change anymore. It will be triggered automatically if you are in the correct state and call the correct command.
- There is still a way to force the state change: “self.state_model._set_obs_state(xxx)”. But be careful with this command, it seems that the device has to be called in the correct state to make the state change.
- To read the current observing state: “Self.state_model._obs_state”

10.5.5.2 Creating a new Command Class:

10.5.5.2.1 New command with the same name

Normally, if the command name is the same as the base class command name, then just inherit from the base command, and implement the do() hook. Be careful if you want to call super().do() or not. I suggest go to the base class and investigate what do() hook does. Most of the time super().do() is not useful. Sometimes it causes problems because the base class SKASubarray implements a resource manager, while Mid-Cbf-MCS doesn't.

```
class OnCommand(SKASubarray.OnCommand):
    """
    A class for the SKASubarray's On() command.
    """
    def do(self):
        """
        Stateless hook for On() command functionality.

        :return: A tuple containing a return code and a string
            message indicating status. The message is for
            information purpose only.
        :rtype: (ResultCode, str)
        """
        (result_code,message)=super().do()
        device = self.target
        device._proxy_sw_1.SetState(tango.DevState.DISABLE)
        device._proxy_sw_2.SetState(tango.DevState.DISABLE)
        return [result_code,message]
```

Figure 10-2 Creating a new command with the same name as in the LMC base class

10.5.5.2.2 New Class with different names

However, Mid-Cbf-MCS has several commands with different names from the command names from the ADR-8 diagram(https://confluence.skatelescope.org/pages/viewpage.action?pageId=105416556)
This is also described in chapter 6 of this document describing subarray operational states.
In this case, in order to implement a different name, you need to:
1. Create a command class

```
class AddReceptorsCommand(SKASubarray.AssignResourcesCommand):
    # doesn't inherit SKASubarray._ResourcingCommand because will give error on len(self.target)
    """
    A class for CbfSubarray's AddReceptors() command.
    """
    def do(self, argin):
        """
        Stateless hook for AddReceptors() command functionality.

        :param argin: The receptors to be assigned
        :type argin: list of int
        :return: A tuple containing a return code and a string
            message indicating status. The message is for
            information purpose only.
        :rtype: (ResultCode, str)
        """
        device=self.target
        # Code here
        errs = []  # list of error messages
        receptor_to_vcc = dict([*map(int, pair.split(":"))] for pair in
                               device._proxy_cbf_master.receptorToVcc)
        for receptorID in argin:
            try:
                vccID = receptor_to_vcc[receptorID]
```

Figure 10-3 Creating a New Command Class

2. Register the command class in “init_command_objects”.

```
    # PROTECTED REGION ID(CbfSubarray.class_variable) ENABLED START #
    def init_command_objects(self):
        """
        Sets up the command objects. Register the new Commands here.
        """
        super().init_command_objects()
        device_args = (self, self.state_model, self.logger)
        # resource_args = (self.resource_manager, self.state_model, self.logger)
        # only use resource_args if we want to have separate resource_manager object

        self.register_command_object(
            "Configure",
            self.ConfigureCommand(*device_args)
        )
        self.register_command_object(
            "AddReceptors",
            self.AddReceptorsCommand(*device_args)
        )
        self.register_command_object(
            "RemoveReceptors",
            self.RemoveReceptorsCommand(*device_args)
        )
        self.register_command_object(
            "RemoveAllReceptors",

LEMS 104  OUTPUT  TERMINAL  DEBUG CONSOLE
```

Figure 10-4 Registering the command class in “init_command_objects”

3. Call the command. Remember to set dtype_out, since it will return the result message with the new base class scheme.

```
@command(
    dtype_in=('uint16',),
    doc_in="List of receptor IDs",
    dtype_out='DevVarLongStringArray',
    doc_out="(ReturnType, 'informational message')"
)
def AddReceptors(self, argin):
    """
    Assign Receptors to this subarray.
    Turn subarray to ObsState = IDLE if previously no receptor is assigned.
    """
    command = self.get_command_object("AddReceptors")
    (return_code, message) = command(argin)
    return [[return_code], [message]]
```

Figure 10-5 Get the command class object in functions

10.6 Publishing image

10.6.1 Change version number

To publish the image, first change the image version in the **".release"** file.
You have to change the version number and release number.

10.6.2 Publish the image

Then push the code onto Gitlab, wait for the CI to finish. On the status bar, there will be an option to publish image. It is indicated as manual work. Click it to publish.
Alternative way: publishing the image from terminal:
Tag the image for the version:

```
git tag –a <midcbf-prototype-0.4.1>
```

Remove the lastest image to be safe:

```
docker image ls

docker image rm –f <name of the latest image>
```

build the image:

```
make build
```

you will see in the build message: mid-cbf-mcs:<version-number>-<git ash>
Check to make sure the version is correct.

Then login to nexus and push:

```
$ docker login --username ci-cd nexus.engageska-portugal.pt
```

It will ask you for password, please ask the system team on slack for it.
Finally push the image:

```
$ docker push nexus.engageska-portugal.pt/ska-docker/mid-cbf-mcs:0.4.2-a67a9cb
```

10.7 Updating SKA MPI

Integration test is done by the NCRA team. The following is an example of the updates needed in SKAMPI when the new image is released. You should check with the NCRA team again before doing so.

10.7.1 Configuration JSON

This is an example file that needs to be updated if there is change for any device specific configuration or if any new device is added/deleted.

https://gitlab.com/ska-telescope/skampi/-/blob/ss-41_multiscan/charts/skampi/charts/cbf-proto/data/midcbfconfig.json

10.7.2 Image version

This is an example file including the image version for Mid-CBF-MCS. You should change it after pushing the new image onto the SKA server.

https://gitlab.com/ska-telescope/skampi/-/blob/ss-41_multiscan/charts/skampi/charts/cbf-proto/values.yaml.

10.7.3 The test cases

This is an example folder where you should update the test cases. The three subfolders acceptance, bugs and smoke need to be updated if there are any specific changes in Mid-CBF, for example: there is a change in CBF device attribute name and that attribute is used in the test cases.

https://gitlab.com/ska-telescope/skampi/-/tree/ss-41_multiscan/post-deployment/tests

10.8 Interacting with devices in Linux shell

The following is an example on how to act like a client without using a GUI. By running the following command as an example, we are acting like a TANGO client.

```
$ Make interactive
$ Python
$ Import tango
$ proxy=tango.DeviceProxy("mid_csp_cbf/sub_elt/subarray_01")
$ proxy.healthState
$ proxy.state()
$ proxy.On()
$ proxy.AddReceptors([1,2])
$ proxy.receptors
$ proxy.state()
```

10.9 Studying Configuration Files

This section describes some important configuration files in the repository.

10.9.1 Makefile:

Make file links all other configuration files together. It specifies instructions like make up/make down/make test. Make file calls docker compose, which read .yml files

10.9.2 Other Files:

"Tango.yml":
Creates containers for Tangodb. Tango images are provided by the system team.

"Mid-cbf-mcs.yml":

- Create containers for midcbf project
- Image created by make build
- Contains command to activate each container, for example: python ../../VccMulti.py vcc001
- Calls midcbf_dsconfig.json to configure tango database

"Midcbf_dsconfig":

- This file is in "./tangods/data"
- It is used for configuring the tango database. It contains the TANGO property information for each tango device.

"Dockerfile":

- Images by the system team to create environment for the project

10.9.2.1 Adding a new device for Mid-CBF-MCS

To do that you have to change both "midcbfmcs.yml" to start container and "dsconfig.JSON" to configure TANGO property, as mentioned above.

11 IP Libraries and Library Management

This chapter is not applicable to the Mid.CBF MCS; no 3rd party IP is used.

12 Software Variations and Management

This chapter is not applicable to the Mid.CBF MCS; only one variation of this software exists.

13 Test Plan

The Mid.CBF MCS was developed as part of the MVP, the end goal of the Evolutionary Prototype for Program Increment #3. As such, this software is not at the point of maturity to justify creating a test plan for pre-production and production.

13.1 Development Test Plan

All testing during development is automated. The different components of the Mid.CBF MCS are unit tested in isolation where possible. Integration tests between the components are necessary to test message flow and state transitions during sequences such as scan configuration or resource allocation.
A TANGO Device is present to simulate TMC and SDP functionality. This device is able to provide delay model updates.

13.2 Prototype Test Plan

The Mid.CBF MCS interfaces with the CSP.LMC, TMC, and SDP. System integration testing of these components is largely manual for the MVP. In the future, this will be fully automated.

13.3 New Product Introduction Test Plan

No consideration has been given to testing pre-production software.

13.4 Full Production Test Plan

No consideration has been given to testing production software.

14 Appendix I: Discrepancies From Design At CSP CDR

The following are discrepancies of the Mid.CBF MCS from the original design of the CSP presented at Critical Design Review (refer to AD1 and AD2).

1. TMC no longer accesses CSP Master to assign and release resources on a particular sub-array's behalf. Instead, TMC accesses the CSP Subarray directly to do so. This behavior has been extended to the CSP to Mid.CBF interface; CSP.LMC accesses the Mid.CBF Subarray directly to assign and release resources.
2. The receptor list has been omitted from the set of scan configuration parameters. Receptors must be added to a sub-array prior to configuring a scan, when the Mid.CBF Subarray is in observing state IDLE.
3. Scan ID and frequency band are now required in the set of scan configuration parameters (were previously optional and memorized).
4. A boolean value to enable transient data capture is now required in the set of search window parameters (was previously optional and defaulted to false). Additionally, destination addresses for transient data are now also required if transient data capture is enabled. This is to facilitate a timestamp, instead of a set of addresses, as the input argument to a call to `offloadTransientDataCapture` (not implemented for the MVP).
5. The CSP TelState device is no longer present nor required. Attributes that were previously exposed on CSP TelState are now exposed directly on CSP Subarray.

15 Appendix II: JSON Examples

This appendix contains examples of JSON objects that are relevant to configuring and executing a scan.

15.1 Scan Configuration

This section provides an example for a JSON script passed via the command Configure().

The example defined the subarray configuration as follows:

- Band 1 observation.
- Use FSP #4 to correlate a Frequency Slice #1.
- Correlate the full FS bandwidth (200 MHz); correlated bandwidth is calculated as:

  200MHz / 2^0 (in the script the correlated bandwidth is specified as zero).
- Integration time is 140 miliseconds (integrate output products for 140 milliseconfs before transmiting to SDP).
- The offset for the channel IDs inserted in the output products is zero, meaning that the ChannelID generated by FSP #4 are in range [0 .. 14788].
- The channel averaging factor is specified per group of 744 channels produced on the same FPGA, in this example the maximum averaging factor (8) is specified for the first group of 744 channels (channels 0 to 743). The averaging factor of zero for other groups means 'do not transmit'; meaning that only the channels 0 to 743, averaged by factor of 8, will be transmitted to SDP.
- The output link map is specified so that channels 0 to 399 are transmitted to SDP via the output link #1 and channels with the Channel ID 400 and up via the link #2.

Note:

1. The receptors are assigned to a subarray in advance of the scan configuration, using the command AssignReceptors(). Optionally a user can specify a subset of receptors to be correlated (not provided in this example).
2. Scan ID is passed by the command StartScan().

```
{
  "id”: “Simple Band1 example, correlation, one FS”,
  "frequencyBand": "1",
  "fsp": [
    {
      "fspID": 4,
      "functionMode": "CORR",
      "frequencySliceID": 1,
      "corrBandwidth": 0,
      "integrationTime": 140,
      "fspChannelOffset": 0,
      "channelAveragingMap": [
        [1, 8],
        [745, 0],
        [1489, 0],
        [2233, 0],
        [2977, 0],
        [3721, 0],
        [4465, 0],
        [5209, 0],
        [5953, 0],
        [6697, 0],
        [7441, 0],
        [8185, 0],
        [8929, 0],
        [9673, 0],
        [10417, 0],
        [11161, 0],
        [11905, 0],
        [12649, 0],
        [13393, 0],
        [14137, 0],
      ],
       “outputLinkMap”: [
         [0,1],
         [400,2],
       ],
       “outputHost”: [[0,”192.168.0.1”],[400, “192.168.0.2]],
       “outputPort”: [[0, 9000, 1], [400, 9000,1]]
    }
  ]
}
```

15.2 Delay Models

The following JSON object is an example of a delay model, published to the subscription point given in scan configuration, that can be updated at any time after a successful scan configuration and during a scan.

```
{
  "delayModel": [
    {
      "epoch": "0",
      "delayDetails": [
        {
          "receptor": 1,
          "receptorDelayDetails": [
            {
              "fsid": 1,
              "delayCoeff": [0.4, 1.2, 2.1, 3.4, 4.7, 5.0]
            },
            {
              "fsid": 2,
              "delayCoeff": [0.4, 1.2, 2.1, 3.4, 4.7, 5.0]
            }
          ]
        },
        {
          "receptor": 2,
          "receptorDelayDetails": [
            {
              "fsid": 1,
              "delayCoeff": [0.4, 1.2, 2.1, 3.4, 4.7, 5.0]
            },
            {
              "fsid": 2,
              "delayCoeff": [0.4, 1.2, 2.1, 3.4, 4.7, 5.0]
            }
          ]
        }
      ]
    }
  ]
}
```