JTRS SINCGARS MAC API Service Definition

V1.0
December 15, 2000

Prepared for the
Joint Tactical Radio System (JTRS) Joint Program Office

Prepared by the
Modular Software-programmable Radio Consortium
Under Contract No. DAAB15-00-3-0001

Revision Summary

1.0	Initial release

Table of Contents

1 INTRODUCTION 1

1.1 OVERVIEW 1
1.2 MAC LAYER DESCRIPTION 2
1.3 MODES OF SERVICE 2
1.4 SERVICE STATES 2
1.5 REFERENCED DOCUMENTS 2

2 UUID 2

3 SERVICES 2

3.1 MAC COMMON UTILITY 4
3.1.1 activateChannel Service. 6
3.1.2 getRFPowerControl Service 6
3.1.3 setRFPowerControl Service 6
3.1.4 getMinTU Service. 6
3.1.5 getMaxTU Service. 6
3.1.6 startScan Service. 6
3.1.7 stopScan Service. 6
3.1.8 clearPreset Service. 7
3.1.9 clearAllPresets Service 7
3.1.10 config/query Service. 7
3.2 TRANSEC SERVICE GROUP. 7
3.2.1 loadFill Service. 9
3.2.2 readFillID Service. 9
3.2.3 loadSeed Service. 9
3.2.4 readSeed Service. 9
3.2.5 zeroize Service. 9
3.2.6 TRANSEC Service Group States. 9
3.3 CHANNELERRORCONTROL SERVICE (N/A). 9
3.4 CHANNELACCESS SERVICE. 10
3.4.1 setAccessParameters Service. 11
3.4.2 setCSMAIGAP Service 11
3.4.3 setCSMAPrecedence. 11
3.4.4 ChannelAccess States 12
3.5 MACADDRESSING SERVICE 13
3.6 DROPCAPTURE SERVICE. 14
3.6.1 dropCapture Service. 14
3.7 QUALITY OF SERVICE (N/A) 15
3.8 MAC REAL TIME PACKET (INHERITED) 16
3.8.1 Parameters filled in by inheriting class. 18

4 SERVICE PRIMITIVES 20

4.1 MAC COMMON UTILITY SERVICE PRIMITIVES. 20
4.1.1 activateChannel Service. 20

4.1.2 getRFPowerControl Service 20
4.1.3 setRFPowerControl Service. 21
4.1.4 getMinTU Length Service 22
4.1.5 getMaxTU Length Service. 23
4.1.6 startScan Service. 23
4.1.7 stopScan Service. 24
4.1.8 clearPreset Service. 24
4.1.9 clearAllPresets Service 25
4.1.10 config/query Name-Value Pairs. 25
4.2 TRANSEC SERVICE PRIMITIVES. 26
4.2.1 loadFill Service. 26
4.2.2 readFillID Service. 27
4.2.3 loadSeed Service. 28
4.2.4 readSeed Service. 30
4.2.5 zeroize Service. 31
4.3 CHANNEL ERROR CONTROL SERVICE PRIMITIVES. 31
4.3.1 setChannelErrorControl Service 31
4.4 CHANNEL ACCESS SERVICE PRIMITIVES. 31
4.4.1 setAccessParameters Service. 31
4.5 MAC ADDRESS SERVICE. 32
4.5.1 Bind Address Service. 32
4.6 DROP CAPTURE SERVICE PRIMITIVES 33
4.6.1 dropCapture Service 33
4.7 QUALITY OF SERVICE(QOS) PRIMITIVES. 34
4.7.1 Read QOS Service 34
4.8 MAC REAL TIME PACKET SERVICE (INHERITED). 34
4.8.1 getMaxPayloadSize 34
4.8.2 getMinPayloadSize 35
4.8.3 SINCGARSMACDownStreamProvider Service. 35
4.8.4 SINCGARS MAC Down Stream Provider Queue Service 35
4.8.5 SINCGARSMACUpStreamUserQueue Service 39

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES. 44

6 UTILIZATION OF SCA BUILDING BLOCKS. 44

7 PRECEDENCE OF SERVICE PRIMITIVES 44

8 SERVICE USER GUIDELINES. 44

9 SERVICE PROVIDER-SPECIFIC INFORMATION. 44

10 IDL. 45

10.1 COMMON INTERFACES. 45
10.2 COMMON TYPES 48
10.3 NON-REAL TIME. 50
10.4 REAL-TIME. 55

11 UML. 58

List of Figures

Figure 1-1. Service Definition Overview 1
Figure 3-1. SINCGARS MAC Common Utility Service 5
Figure 3-2. SINCGARS TRANSEC Service Group 8
Figure 3-3. Channel Error Control Building Block 9
Figure 3-4. SINCGARS Channel Access Service 10
Figure 3-5. Sequence Diagrams, setAccessParameters 11
Figure 3-6. State Diagram, ChannelAccess 12
Figure 3-7. MACAddressing (N/A) 13
Figure 3-8. SINCGARS Drop Capture Service 14
Figure 3-9. SINCGARS Sequence Diagram, dropCapture 14
Figure 3-10. Quality of Service (N/A) 15
Figure 3-11. SINCGARS Real Time Packet 17

List of Tables

Table 1. Cross-Reference of SINCGARS Services and Primitives 3

1 INTRODUCTION.

1.1 OVERVIEW.

The SINCGARS Media Access Control (MAC) application-program interface (API) provides a standardized interface to the MAC Layer. Location of the MAC Layer with respect to other JTRS layers is shown in Figure 1-1.

The MAC Services are grouped into non-real time and real time services. The MAC non-real time signals go through the "B" interface while the real time signals go through the "A" interface. In reality, these are the same interface, but it helps conceptually to think of them as different interfaces based on the type of service provided.

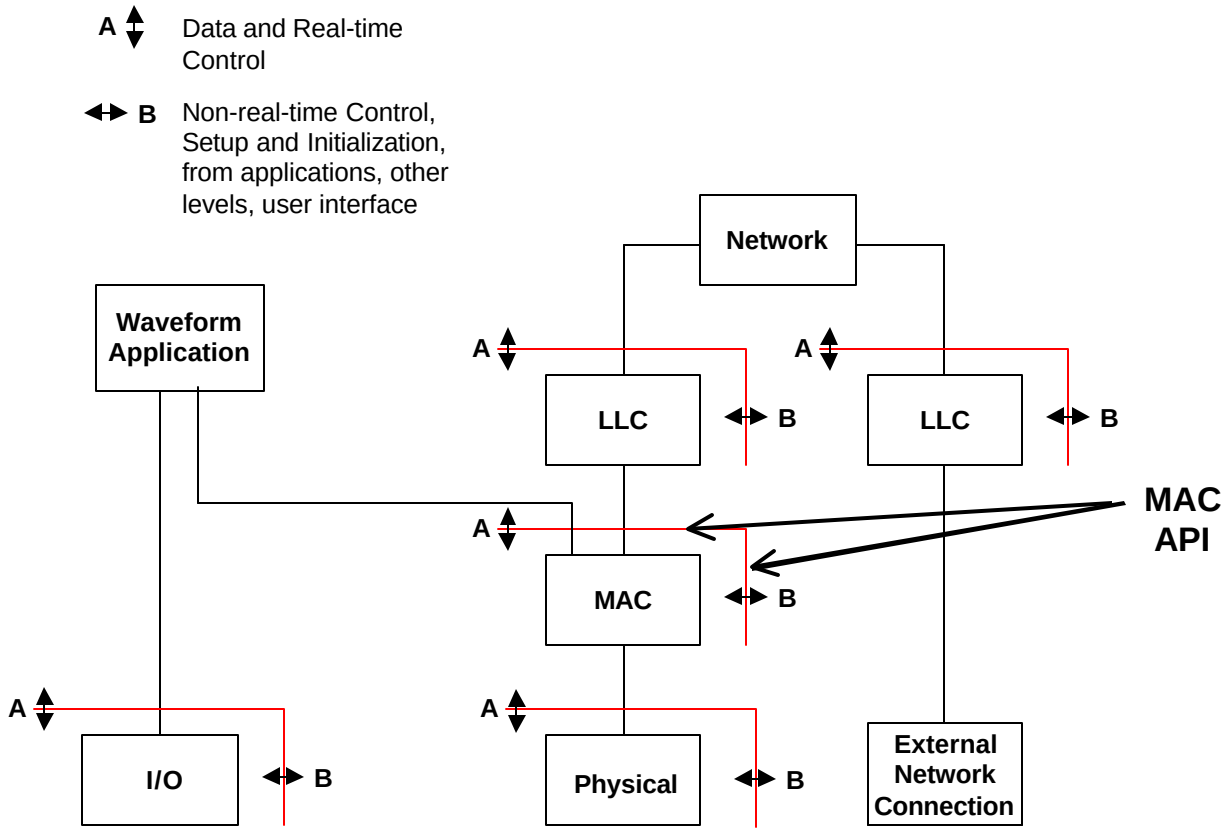

Figure 1-1. Service Definition Overview

The non-real time part of the MAC API consists of instantiations of JTRS MAC Building Blocks (BBs) into concrete classes for SINCGARS. The real time part of the MAC API is an instantiation of the JTRS Packet Building Block.

Note that in the current version of this document, exceptions are not fully specified. This should not be interpreted to mean that the methods cannot raise exceptions. Exceptions will be defined in a later revision to conform to SCA requirements and good SW Engineering practice.

1.2 MAC LAYER DESCRIPTION.

The primary Server used by the MAC Layer client is the Physical Layer. The MAC Service Layer is subdivided into the following Service Groups:

- MAC Common Utility
- TRANSEC
- Channel Error Control
- Channel Access
- MAC Address
- Drop Capture
- Quality of Service (QOS)
- Packet (Provided by the MAC instantiation of the Packet API).

1.3 MODES OF SERVICE.

The MAC API is used for radio configuration, and in transmit and receive modes.

1.4 SERVICE STATES.

The MAC Service State is dependent on the SINCGARS waveform mode.

1.5 REFERENCED DOCUMENTS.

Document No.	Document Title
MSRC-5000SCA	Software Communications Architecture Specification
A3191133	Specification – Mode 2 and Mode 3 Fill, Interface for
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix C Generic Packet Building Block Service Definition

2 UUID.

The UUID for this API is `69695c60-d1d3-11d4-8cc8-00104b23b8a2`.

3 SERVICES

The Services and Primitives provided by the non-real time SINCGARS MAC Layer are shown in the following table.

Table 1. Cross-Reference of SINCGARS Services and Primitives

Service Group	Service	Primitives
MAC Common Utility	Activate Channel	activateChannel (PresetNum : in short) : boolean [1]
	Get RF Power Control	getRFPowerControl(PowerMode: out SINCGARSPowerModeType) : void
	Set RF Power Control	setRFPowerControl (PowerMode : in PowerModeType) : boolean [1]
	Get Minimum Tx Unit Length	getMinTU (MinTU : out unsigned long) : void [1]
	Get Maximum Tx Unit Length	getMaxTU (MaxTU : out unsigned long) : void [1]
	Single Channel Freq Scan	startScan( ) : void
		stopScan( ) : void
	Clear One Preset	clearPreset(PrestNum : in short) : void
	Clear All Presets	clearAllPresets( ) : void
TRANSEC	Load Fill	loadFill (presetNum : in short, Fill : in SINCGARSFillType, FillInfo : in SINCGARSFillInfoType) : boolean [1]
	Set Fill ID	Uses CF::Resource:: Config/Query
	Read Fill ID	readFillID (PresetNum : in short, Fill : in SINCGARSFillType, FillID : out SINCGARSFillIDType) : boolean [1]
	Load TOD Seed	loadSeed (SeedNum : in short, Seed : in SINCGARSSeedType, SeedInfo : in SINCGARSSeedInfoType) : boolean [1]
	Read TOD Seed	readSeed (SeedNum : in short, Seed: in SINCGARSSeedType, SeedInfo : out SINCGARSSeedInfoType): boolean [1]
	Zeroize Fill/Seed	zeroize ( ) : boolean [1]
Channel Error Control	Enable/Disable Error Control	(Building Block not used in SINCGARS)

Table 1. Cross-Reference of SINCGARS Services and Primitives - Continued

Service Group	Service	Primitives
Channel Access	Set CSMA Parameters	setAccessParameters(Parameters : in SINCGARSAccessParameterListType) : boolean [1]
	Set CSMA Intertransmission Delay	SetCSMAIGAP(IGAP : in short) : void
	Set CSMA Precedence	SetCSMAPrecedence(precedence : in short) : void
MAC Address	Bind Own Address	Uses CF::Resource:: Config/Query
Drop Capture	Enable/Disable	dropCapture () : boolean
QOS	Read QOS	(Building Block not used in SINCGARS)
	Set QOS Parameters	
	Enable QOS	
	Disable QOS	

[1] – Primitives defined by MAC Building Block

3.1 MAC COMMON UTILITY.

The MAC Common Utility Service Group (Figure 3-1) provides the interface for setting the operational functionality of the MAC Layer. This interface provides for independent selection of:

- activate channel – e.g., net selections, manual, cue
- set RF power level – e.g., High, Medium, Low, Power Amplifier
- scan frequency control – e.g., start and stop
- clear all presets.

Other MAC parameters listed in the note below are set and read using the *config/query* interface from the Core Framework.

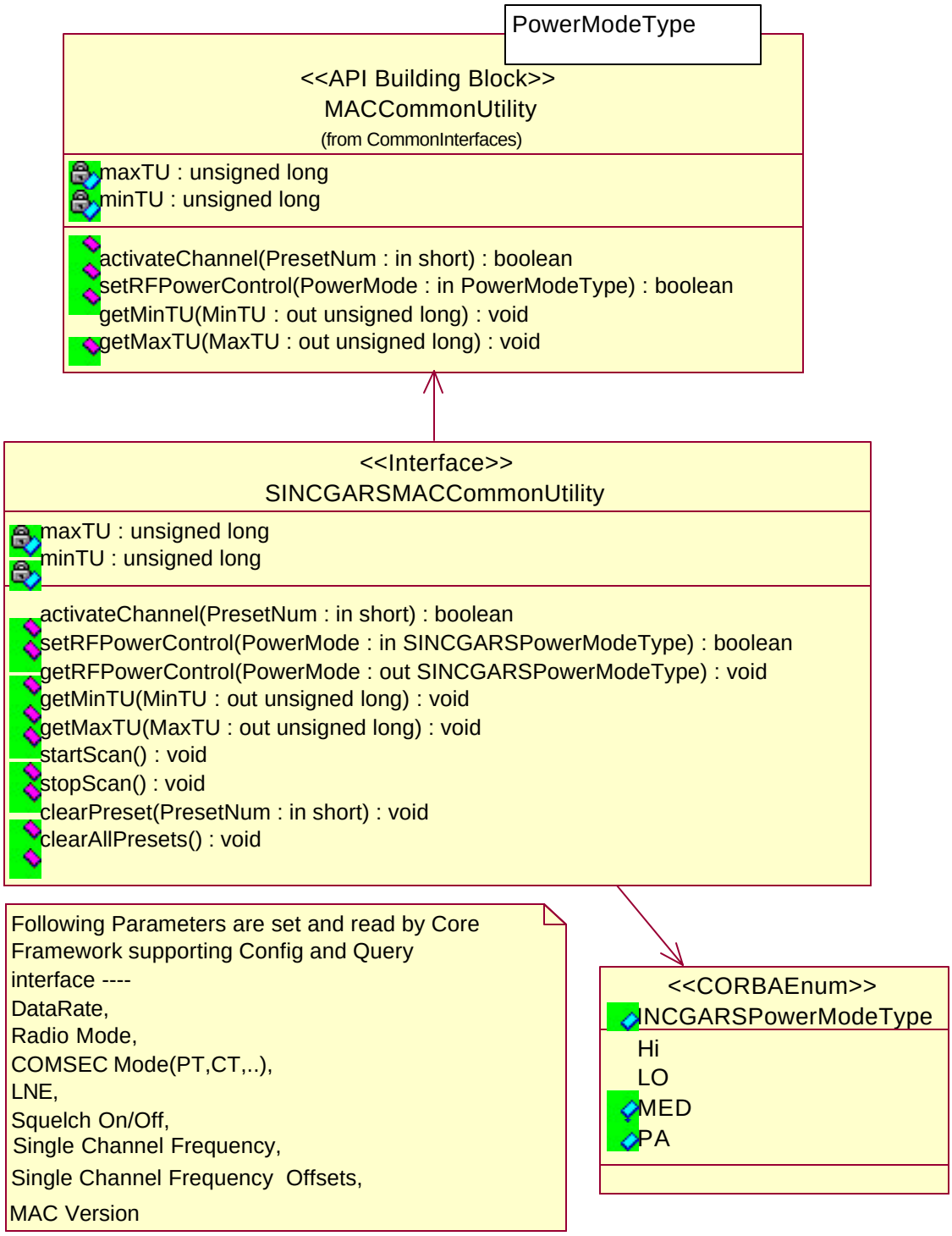

Figure 3-1. SINCGARS MAC Common Utility Service

3.1.1 activateChannel Service.

The activateChannel Service provides Service Users with a method to communicate to the MAC Layer the active radio channel (e.g. 0, 1, 2…6, where 0 = manual, 7 = CUE).

Typically, this service is used by the Human Control Interface to allow a radio operator to designate a Preset Channel to be the operational channel. When the Preset Number specified by the Service User is a channel number known by the Common Utility Server, the boolean returned by the method indicates *Channel Activated* (i.e., TRUE). Otherwise, it indicates *Preset Number Unknown or Preset not filled* (i.e., FALSE).

3.1.2 getRFPowerControl Service.

getRFPowerControl returns the current transmit power level selection as High, Low, Medium, PA, or Radio Silence.

3.1.3 setRFPowerControl Service

setRFPowerControl Service provides Service Users with a method to communicate to the MAC Layer the selected RF Power Control mode (e.g. Hi, Lo, Med, PA). Typically, this service would be used by the Human Control Interface to allow a radio operator to designate the RF power mode for the operational RF Channel. When the *RFPowerMode* specified by the Service User is a power mode known by the Common Utility, the boolean returned by the method indicates *RF Power Mode Set* (i.e., TRUE). Otherwise, it indicates Power *RF Mode Unknown* (i.e., FALSE).

3.1.4 getMinTU Service.

getMinTU Service returns the minimum number of Transmission Units (TUs) that will be sent in one over-the-air transmission by the Physical Layer. This is a read-only method. The SINCGARS Transmission Unit is TBD. The minimum number of Transmission Units for SINCGARS is TBD.

3.1.5 getMaxTU Service.

getMaxTU Service returns the maximum number of Transmission Units that will be sent in one over-the-air transmission by the Physical Layer. This is a read-only method. The SINCGARS Transmission Unit is TBD. The maximum number of Transmission Units for SINCGARS is TBD.

3.1.6 startScan Service.

startScan Service is available in single channel operation. This service starts scanning the single channel frequencies associated with the presets. Note that scan functionality is not completely defined by the inherited Building Block operations.

3.1.7 stopScan Service.

stopScan Service stops the single channel frequency scan function, if active. If not active, this method has no effect.

3.1.8 clearPreset Service.

clearPreset Service causes all parameters associated with the selected preset to be zeroized (i.e. single channel frequency(ies), hopset, hopset ID, net time-of-day, etc.).

3.1.9 clearAllPresets Service.

clearAllPresets Service causes all preset parameters to be zeroized (i.e. single channel frequencies, hopsets, hopset IDs, net time-of-day's, etc.).

3.1.10 config/query Service.

The config/query services are inherited from *Resource* in Core Framework. The following MAC parameters are set and/or read using this service.

- Data Rate
- Radio Mode
- Comsec Mode(PT, CT,…)
- Late Net Entry
- squelch on/off
- Single Channel frequency
- Single Channel frequency offsets
- MAC Version.

3.2 TRANSEC SERVICE GROUP.

The SINCGARS Transmission Security (TRANSEC) Service Group (Figure 3-2) provides a generalized TRANSEC interface to the MAC Layer. The TRANSEC function generates pseudorandom numbers used for frequency selection and data overlays. Hopset ID and SINCGARS Time Seed are required by the algorithm. This Service group also provides methods for transferring hopsets and lockouts into the MAC Layer from the INFOSEC. The zeroize method allows the user to zeroize all TRANSEC data that has been loaded into the MAC Layer.

Note, not all TRANSEC services may be externally visible for a given implementation of SINCGARS (for example if the TRANSEC is implemented in the modem, no real-time control may be available).

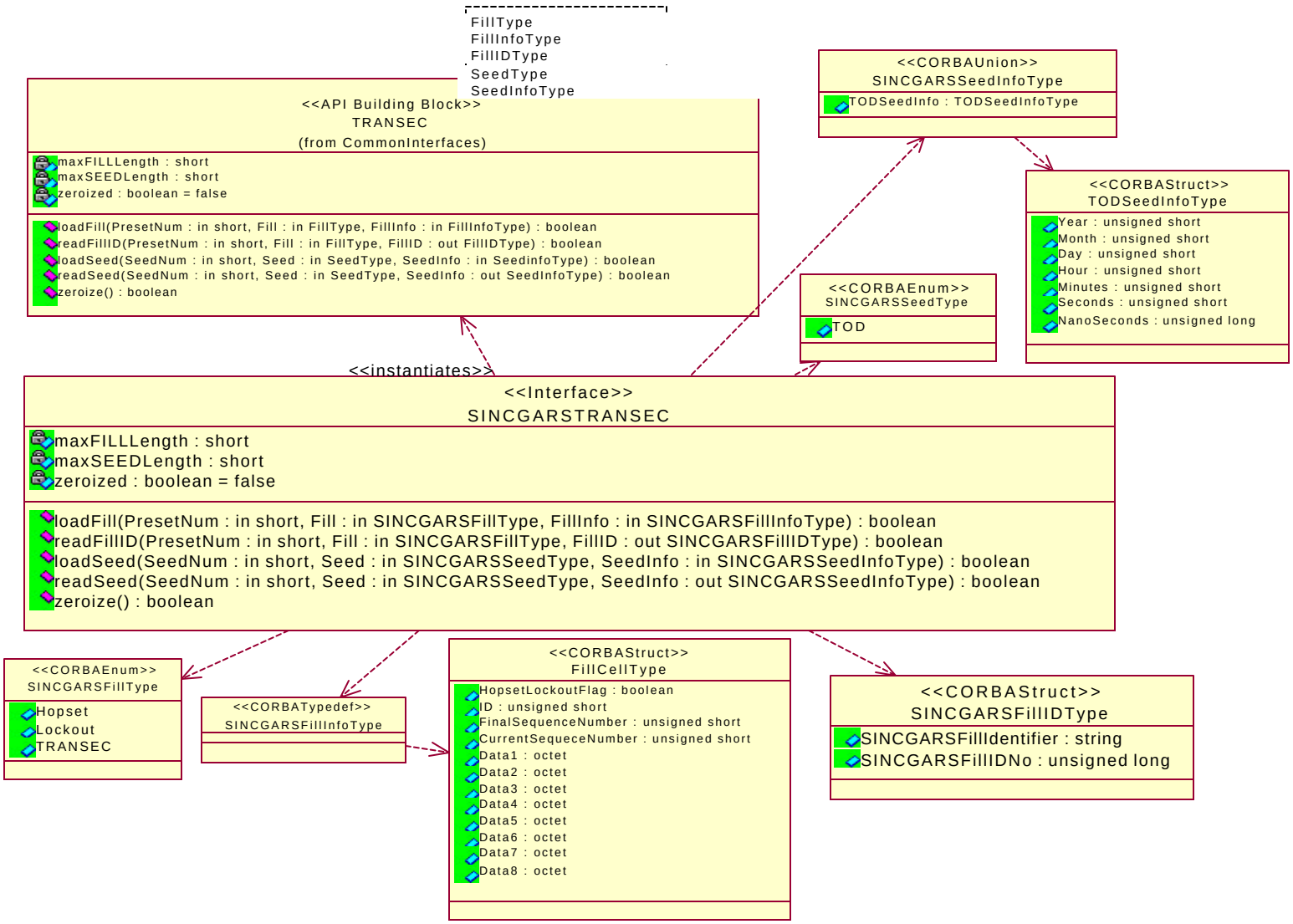

Figure 3-2. SINCGARS TRANSEC Service Group

3.2.1 loadFill Service.

The SINCGARS loadFill Service provides a method to transfer TRANSEC Fill data from the Service User to the MAC Layer, but never in the reverse direction. It returns the status of each transfer, good or bad. SINCGARSFillInfoType can be either a Hopset or a Lockout cell.

3.2.2 readFillID Service.

The SINCGARS readFillID Service provides Service Users with a method of reading the identity of a fill element for a preset but not the actual Fill data. As an example, *readFillID* will return a Hopset's (or Lockout's) Fill Identifier and Fill ID number but not the Hopset (or Lockout) data. For a TRANSEC Variable, it will return the variable's ID, but not the variable.

3.2.3 loadSeed Service.

The SINCGARS loadSeed Service provides for transfer of TRANSEC Seed data from the Service User to the MAC Layer. It returns the status of each transfer, good or bad. Seed Type is SINCGARSSeedInfoType which is generated from several TOD parameters.

3.2.4 readSeed Service.

The SINCGARS readSeed Service provides Service Users with a method of reading the current Seed data. As an example, *readSeed* returns current TOD.

3.2.5 zeroize Service.

The SINCGARS zeroize Service provides Service Users with a method of removing all Fill and Seed data from memory space owned by the TRANSEC. This service over-writes all Fill and Seed data memory space with a variety of data patterns to ensure information is erased. This service does not release this memory space back to the Operating Environment.

3.2.6 TRANSEC Service Group States.

3.3 CHANNELERRORCONTROL SERVICE (N/A).

ChannelErrorControl Service is not provided by SINCGARS. Error control is fixed by the operational mode selected (refer to Figure 3-3).

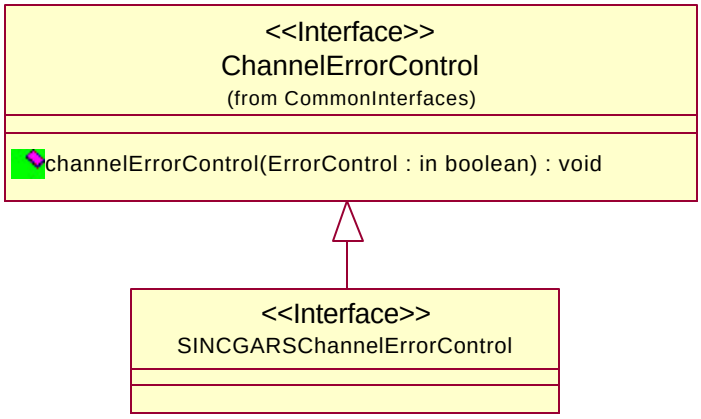

Figure 3-3. Channel Error Control Building Block

3.4 CHANNELACCESS SERVICE.

The ChannelAccess Service Group (Figure 3-4) detects incoming messages and executes the channel access protocol. These functions include:

- sync detection
- sync search
- end-of-message detection
- channel access algorithm – CSMA.

SINCGARS uses the Channel Access Service group to initialize CSMA parameters.

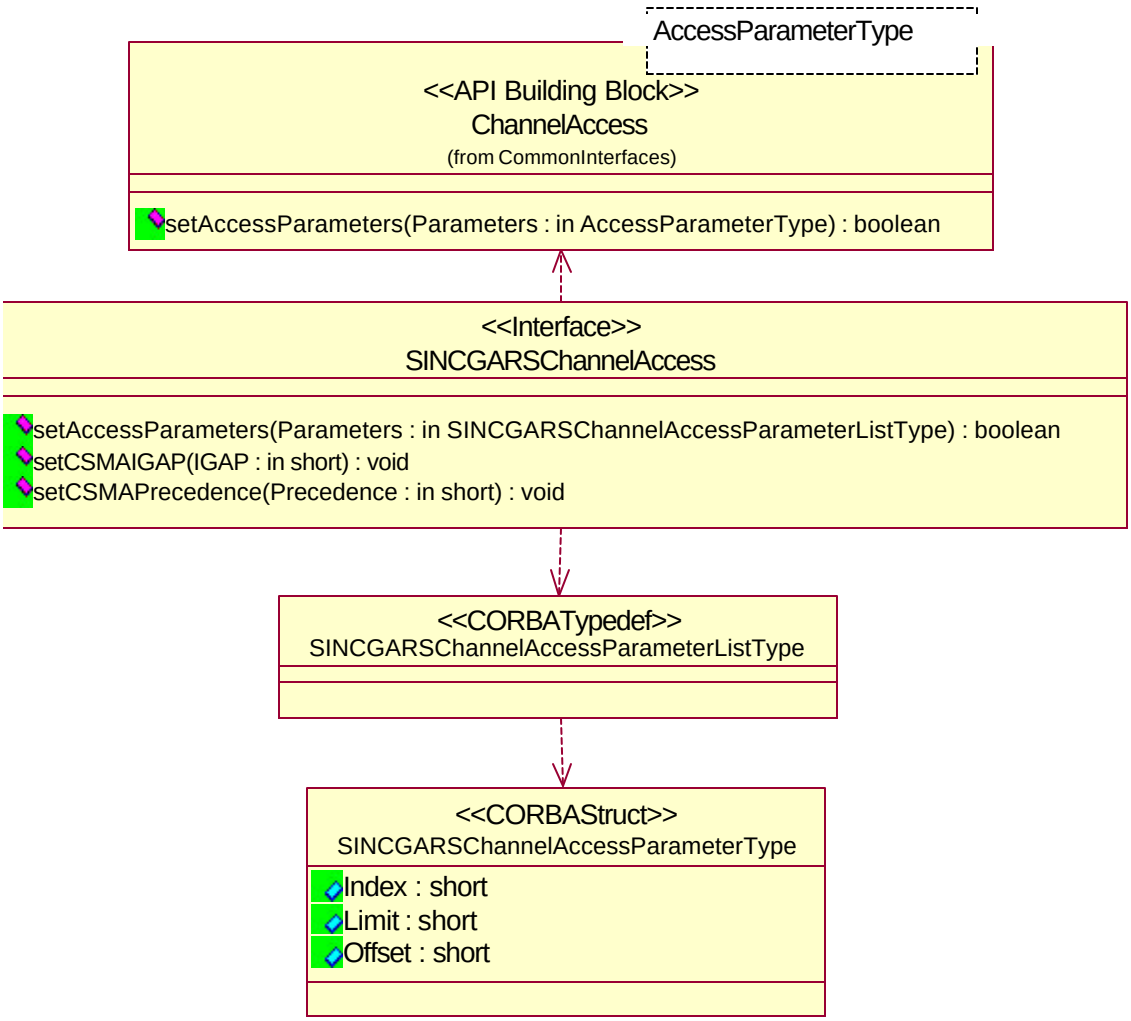

Figure 3-4. SINCGARS Channel Access Service

3.4.1 setAccessParameters Service.

The setAccessParameters Service provides Service Users with the ability to initialize or configure the SINCGARS CSMA parameters index, limit, and offset. Each set configures a single precedence level. Refer to Figure 3-5 for sequencing diagrams.

3.4.2 setCSMAIGAP Service.

setCSMAIGAPServices sets the time interval from the start of receiver idle state following a valid reception to the beginning of the first CSMA contention interval.

3.4.3 setCSMAPrecedence.

This operation changes the CSMA transmit precedence for SINCGARS Data Packets.

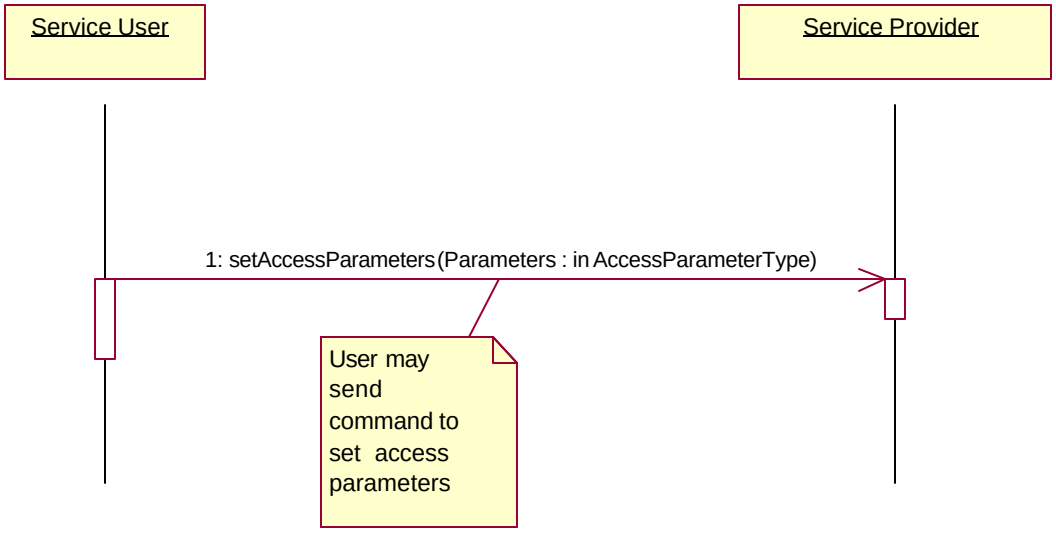

Figure 3-5. Sequence Diagrams, setAccessParameters

3.4.4 ChannelAccess States.

The ChannelAccess Service (Figure 3-6) provides Service Users with controlled access to a RF channel. To achieve controlled access to a particular slot, the access control algorithm must be properly configured (as defined by the instantiating API Service Description). Access control algorithm configuration is achieved via the setAccessParameters method. Otherwise, channel access is immediate. Precedence can be specified on a per-packet basis.

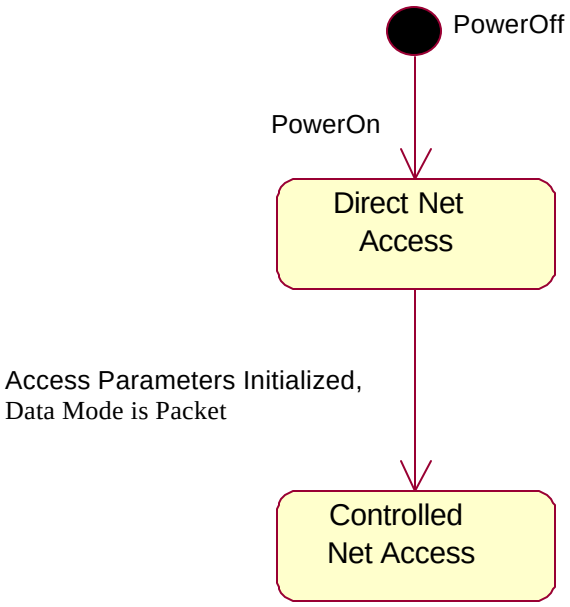

Figure 3-6. State Diagram, ChannelAccess

3.5 MACADDRESSING SERVICE.

The MACAddressing Building Block (Figure 3-7) allows the Service User to define the radio's own intranet address. The SINCGARS intranet address is part of MIL STD 188-220 LLC protocol located on the red (e.g. encrypted) side of the radio disjoint from the MAC Layer on the black (e.g. unencrypted) side of the radio. The SINCGARS intranet address is set using the *config*/*query* interface. SINCGARS does not use this Service Building Block.

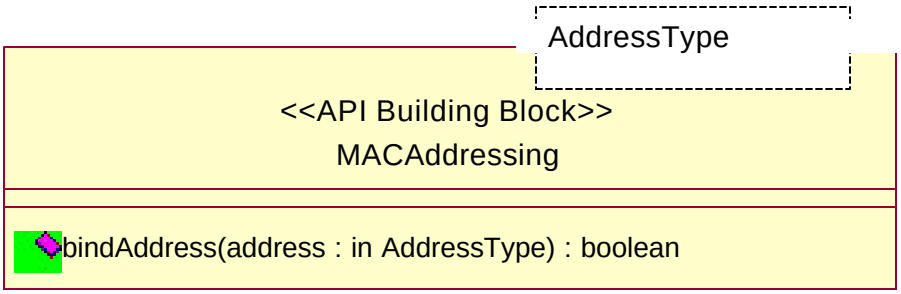

Figure 3-7. MACAddressing (N/A)

3.6 DROPCAPTURE SERVICE.

The DropCapture Service (Figure 3-8) allows a Service User to terminate an active reception and initiate a transmission. The service returns a boolean to indicate reception was terminated (True) or the operation failed (False).

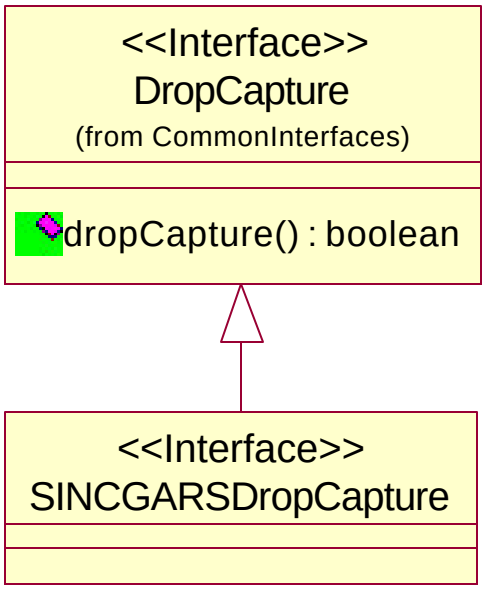

Figure 3-8. SINCGARS Drop Capture Service

3.6.1 dropCapture Service.

dropCapture Service causes the Radio to immediately terminate the current reception and to initiate a transmission. Refer to Figure 3-9 for additional information.

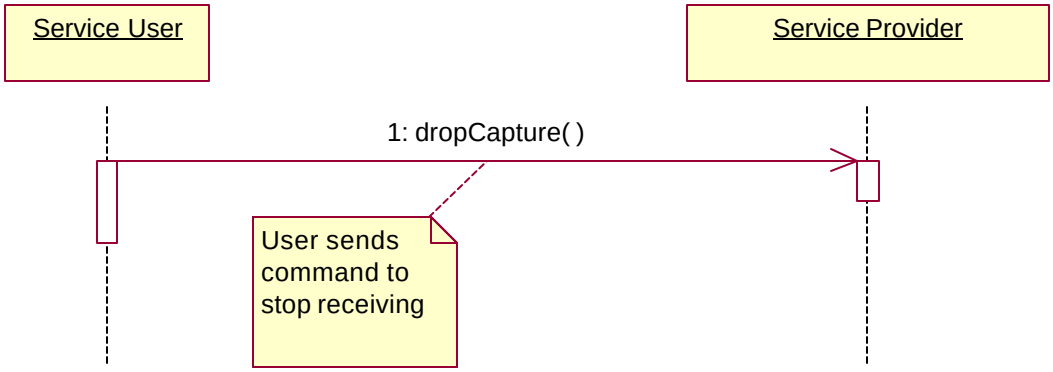

Figure 3-9. SINCGARS Sequence Diagram, dropCapture

3.7 QUALITY OF SERVICE (N/A).

QualityOfService Service (Figure 3-10) provides Service Users with a standard method to obtain channel quality information, when these are calculated in non-real time within the MAC Layer. Examples of these parameters are:

- residual bit-error-rate
- block error rate and
- raw bit-error-rate.

SINCGARS does not provide this non-real time service.

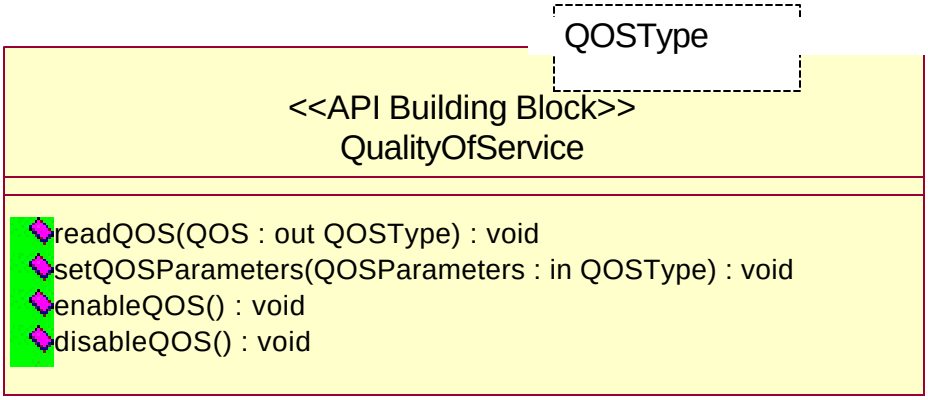

Figure 3-10. Quality of Service (N/A)

3.8 MAC REAL TIME PACKET (INHERITED).

Real-time control and data will be pushed upstream and downstream using control and data packets. The data packets (Figure 3-11) include a control header for transferring real time control information with the data. For more information on the Packet Building Block, refer to the SCA Service Definition Description for the Packet Building Block.

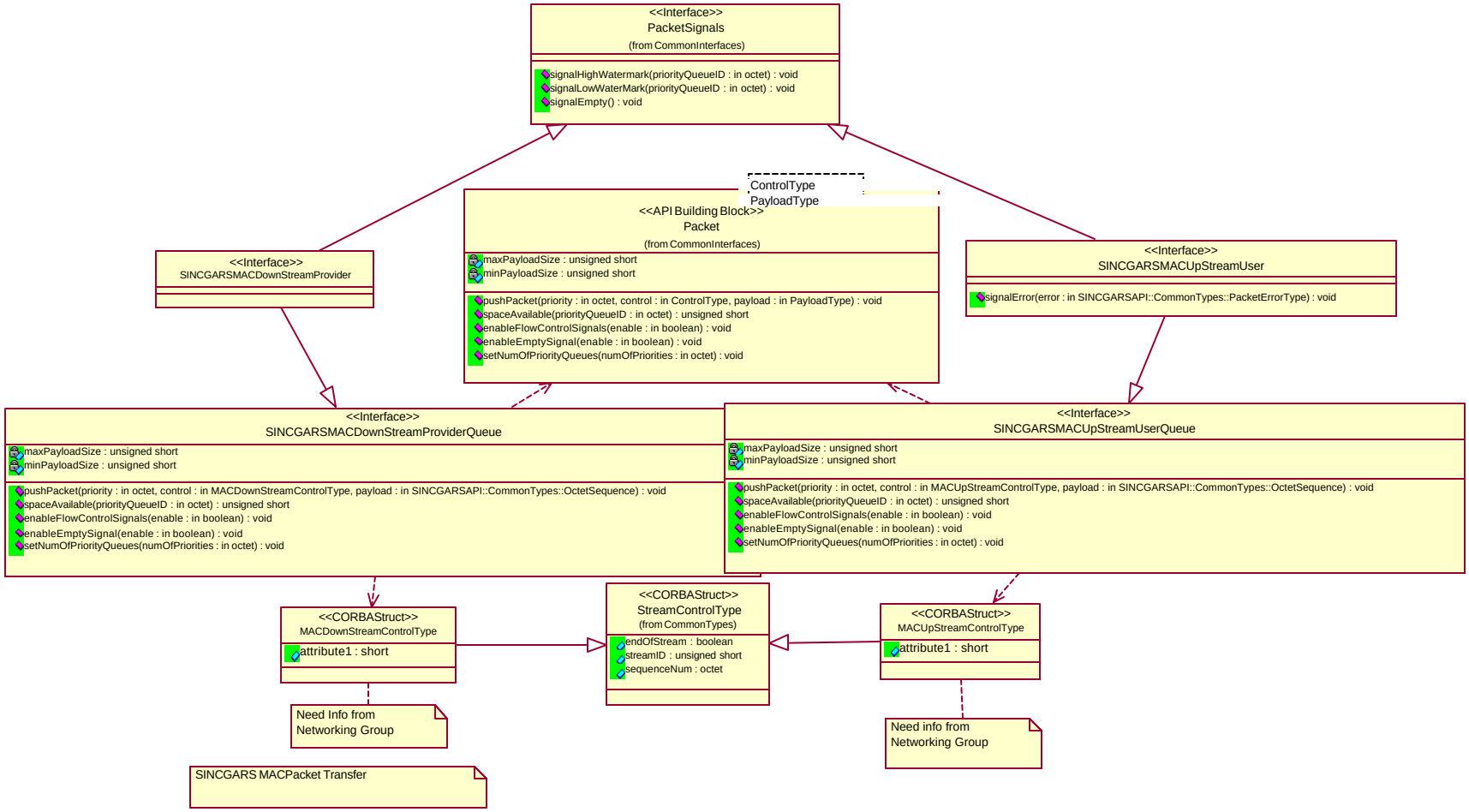

Figure 3-11. SINCGARS Real Time Packet

3.8.1 Parameters filled in by inheriting class.

The following parameters are SINCGARS specific and defined by the instantiating the MAC API Service Definition.

3.8.1.1 maxPayloadSize.

maxPayloadSize defines the maximum size of PayloadType in Traffic Units (TUs). This item allows the Service Provider to bound its internal memory usage. SINCGARS traffic units are in octets. The maximum payload size supported by the downstream MAC Service Provider is **TBD** octets. The maximum payload size supported to the upstream MAC Service User is **TBD** octets. This is a read-only parameter.

3.8.1.2 minPayloadSize

minPayloadSize defines the minimum size of *PayloadType* in Traffic Units. This item allows the Service Provider to bound its internal memory usage. SINCGARS traffic units are in octets. The minimum payload size supported by the downstream MAC Service Provider is **TBD** octets. The minimum payload size supported to the upstream MAC Service User is **TBD** octets. This is a read-only parameter.

3.8.1.3 setNumOfPriorityQueues.

setNumOfPriorityQueues defines the number of different priority queues supported at the MAC Service Provider or the number of different priority queues supported for the MAC upstream user. The number of queues supported by the SINCGARS MAC Service Provider is **TBD**. The number of priority queues supported for the SINCGARS MAC upstream user is **TBD**.

3.8.1.4 signalHighWaterMark.

signalHighWaterMark is determined independently for the upstream and downstream queues by the SINCGARS waveform; it is not settable by the Service User. signalHighWaterMark is valid only after Flow Control Signals have been enabled. signalHighWaterMark is returned from the upstream and downstream queues to the queue user whenever the incoming data causes one of the priority queues to exceed its High Water Mark. The High Water Mark signal also indicates which priority queue has gone above the High Water Mark.

3.8.1.5 signalLowWaterMark.

signalLowWaterMark is determined independently for the upstream and downstream queues by the SINCGARS waveform. It is not settable by the Service User. signalLowWaterMark is valid only after Flow Control Signals have been enabled. signalLowWaterMark is returned from the upstream and downstream queue to the queue user whenever the incoming data causes one of the priority queues to go below its Low Water Mark. The Low Water Mark signal also indicates which priority queue has gone below the Low Water Mark.

3.8.1.6 enableFlowControlSignals.

enableFlowControlSignals enables and disables High Water Mark and Low Water Mark signals.

3.8.1.7 signalEmpty.

signalEmpty is returned to the queue user when either all upstream queues or all downstream queues become empty. This signal is valid only if the Empty Signal is enabled by the enableEmptySignal.

3.8.1.8 *enableEmptySignal.*

enableEmptySignal enables or disables the Empty Signal. SINCGARS supports this control of the Empty Signal on both its upstream and downstream queues independently.

3.8.1.9 spaceAvailable.

spaceAvailable allows the queue user to determine the current space available (in octets) for the requested priority queue. SINCGARS MAC supports this method for both its upstream and downstream queues.

3.8.1.10 pushPacket.

pushPacket is the method to move real time payload and control data into and out of the MAC interface. Downstream is moving data into the MAC Layer (or towards the antenna) while upstream is moving data out of the MAC Layer (or away from the antenna).

3.8.1.10.1 pushPacket payload.

The SINCGARS information payload is the data to be transmitted over-the-air by the Physical Layer or the data received over-the-air by the Physical Layer. The SINCGARS Information payload is in octets. The maximum and minimum number of octets per pushPacket is set by *MaxPayloadSize* and *MinPayloadSize.*

3.8.1.10.2 pushPacket control.

MAC___StreamControlType contains the control attributes used in transferring data to and from the MAC layer. The SINCGARS control attributes are *streamID, SequenceNum,* and *endOfStream.* Several pushPackets can be cancatenated together to form one long data stream using this StreamControlType.

3.8.1.10.2.1 streamID.

Each stream has a different streamID. pushPackets with the same streamID are part of the same stream.

3.8.1.10.2.2 sequenceNum.

sequenceNum is the number of the pushPacket within a stream. The first pushPacket has the sequence number zero. The next pushPacket would have a one (if there are at least two pushPackets in the stream, etc.).

3.8.1.10.2.3 endOfStream.

endOfStream tells the queue that this is the last pushPacket in the stream with this streamID.

3.8.1.11 Packet Error Type.

Packet Error Type for the MAC Layer is inherited from the Logical Link Control layer.

4 SERVICE PRIMITIVES.

This section is in preliminary draft form, and is under review and may be subject to revision.

4.1 MAC COMMON UTILITY SERVICE PRIMITIVES.

4.1.1 activateChannel Service.

activateChannel provides the service user with the capability to the select one of the presets 0,1, 2,…7 (0 = manual, 7 = CUE) at the MAC Layer interface.

4.1.1.1 Synopsis.

```
boolean activateChannel {
        in short        PresetNum
}
```

4.1.1.2 Parameters.

PresetNum

specifies the channel number or preset number of the active (selected) RF channel.

4.1.1.3 State.

Valid in any state.

4.1.1.4 New State.

TBD.

4.1.1.5 Response.

This method returns True if the channel corresponding to the PresetNum is activated; otherwise, False is returned.

4.1.1.6 Originator.

Service User who is typically the HCI.

4.1.1.7 Errors/Exceptions.

Return = True if the channel is successfully activated; otherwise return = False.

4.1.2 getRFPowerControl Service.

getRFPowerControl provides the ability to read the current RF power output level at the MAC Layer interface.

4.1.2.1 Synopsis.

```
Void  getRFPowerControl (
                out SINCGARSPowerModeType          PowerMode
)
```

4.1.2.2 Parameters.

PowerMode

describes the RF output power level of the radio.

```
enum SINCGARSPowerModeType{
        Hi
        Lo
        Med
        PA
}
```

4.1.2.3 State.

Service valid in any state.

4.1.2.4 New State.

No change in state.

4.1.2.5 Response.

Method returns the current output RF level setting.

4.1.2.6 Originator.

This primitive is initiated by the Service User typically the HCI.

4.1.2.7 Errors/Exceptions.

None.

4.1.3 setRFPowerControl Service.

setRFPowerControl provides the ability to communicate the desired RF power output level Low, Medium, High, or PA to the MAC Layer.

4.1.3.1 Synopsis.

```
Boolean  setRFPowerControl {
            in SINCGARSPowerModeType            PowerMode
 }
```

4.1.3.2 Parameters.

PowerMode

selects the RF output power level of the radio.

```
enum SINCGARSPowerModeType{
        Hi
        Lo
        Med
```

PA

}

4.1.3.3 State.

Service valid in any state.

4.1.3.4 New State.

New RF output power level.

4.1.3.5 Response.

Return = True if the power control is successfully set with the given PowerMode, otherwise it Boolean = False.

4.1.3.6 Originator.

This primitive is initiated by the Service User typically the HCI.

4.1.3.7 Errors/Exceptions.

TBD.

4.1.4 getMinTU Length Service.

getMinTU provides Service Users with the minimum Transmission Unit (TU) length the Service Provider will accept as being a valid over-the-air message length..

4.1.4.1 Synopsis.

Void getMinTU {

out unsigned long MinTU

}

4.1.4.2 Parameters.

MinTU

is the minimum length of the data portion (i.e. excluding the control portion) in a valid SINCGARS packet. The Transmission Units are **TBD**. The Minimum number of Transmission Units is **TBD**.

4.1.4.3 State.

This value is fixed at waveform instantiation and does not change.

4.1.4.4 New State.

State remains unchanged.

4.1.4.5 Response.

This is a read-only parameter

4.1.4.6 Originator.

Service User.

4.1.4.7 Errors/Exceptions.

TBD.

4.1.5 getMaxTU Length Service.

getMaxTU provides Service Users with the maximum Transmission Unit (TU) length the Service Provider will accept as being a valid over-the-air message length.

4.1.5.1 Synopsis.

```
Void   getMaxTU{
                    out unsigned long              MaxTU
}
```

4.1.5.2 Parameters.

MaxTU

is the maximum length of the data portion (i.e. excluding the control portion) in a valid SINCGARS packet. The Transmission Units are **TBD**. The Maximum number of Transmission Units is **TBD**.

4.1.5.3 State.

This value is fixed at waveform instantiation and does not change.

4.1.5.4 New State.

State remains unchanged.

4.1.5.5 Response.

This is a read-only parameter

4.1.5.6 Originator.

Service User.

4.1.5.7 Errors/Exceptions.

TBD.

4.1.6 startScan Service.

startScan starts the scanning of all Single Channel frequencies (which are non-zero) in the presets.

4.1.6.1 Synopsis.

```
void  startScan ( )
```

4.1.6.2 Parameters.

None.

4.1.6.3 State.

Service valid in Single Channel **TBD** modes only.

4.1.6.4 New State.

Radio receiver dwells on each single channel frequency for a short period time searching for a valid signal.

4.1.6.5 Response.

The radio enters a single channel frequency scan mode.

4.1.6.6 Originator.

This primitive is initiated by the Service User, typically the HCI.

4.1.6.7 Errors/Exceptions.

TBD.

4.1.7 stopScan Service.

stopScan stops the scanning of all Single Channel frequencies (which are non-zero) in the presets.

4.1.7.1 Synopsis.

```
void  stopScan ( )
```

4.1.7.2 Parameters.

None.

4.1.7.3 State.

Service valid in Single Channel scan mode only.

4.1.7.4 New State.

Radio receiver stops scanning and tunes to the single channel frequency selected by the last *activateChannel* method.

4.1.7.5 Response.

The radio enters single channel receive state.

4.1.7.6 Originator.

This primitive is initiated by the Service User, typically the HCI.

4.1.7.7 Errors/Exceptions.

TBD.

4.1.8 clearPreset Service.

clearPreset allows the Service User to set the parameters stored in the designated preset to zero.

4.1.8.1 Synopsis.

```
void  clearPreset {
                    in short                    PresetNum
}
```

4.1.8.2 Parameters.

PresetNum

is the the preset number 0 to 7 (0 = Manual, 7 = CUE).

4.1.8.3 State.

TBD.

4.1.8.4 New State.

The designated preset contains no valid fill information.

4.1.8.5 Response.

An attempt to transmit using this preset will result in a "no fill" warning to the user.

4.1.8.6 Originator.

This primitive is initiated by the Service User, typically the HCI.

4.1.8.7 Errors/Exceptions.

TBD.

4.1.9 clearAllPresets Service.

clearAllPresets allows the Service User to set the parameters stored in all presets to zero.

4.1.9.1 Synopsis.

void clearAllPresets ( )

4.1.9.2 Parameters.

None.

4.1.9.3 State.

TBD.

4.1.9.4 New State

All presets are void of fill information.

4.1.9.5 Response.

Any attempt to transmit using any preset will result in a "no fill" warning to the user.

4.1.9.6 Originator.

This primitive is initiated by the Service User, typically the HCI.

4.1.9.7 Errors/Exceptions.

TBD.

4.1.10 *config/query* Name-Value Pairs.

The following are the *config/query* name-value Pairs for the MAC Common Utility. This sequence of name-value pairs are set and read using the *config/query* operations inherited from

Resource in the Core Framework. Name value pairs are specified in the format: name, type, value.

Name	Type	Values
DataRate	Enum	600, 1200, 2400, 4800, 16000, 1200N, 2400N, 4800N, 9600N, AD1, TF, Pckt, RS232, Off
RadioMode	Enum	SC, FH, FHM
COMSEC	Enum	PT, CT
LNE	boolean	On, Off
Squelch	boolean	On, Off
SCFreq	long	30.000 mHz to 87.975 mHz
FreqOffset	Enum	0 , +5 kHz, +10 kHz, -5 kHz, -10 kHz
FillID	unsigned short	0 to 999
MACVersion	long	MACVersionUUID

4.2 TRANSEC SERVICE PRIMITIVES.

4.2.1 loadFill Service.

loadFill provides a method to transfer TRANSEC Fill data to the MAC Layer.

4.2.1.1 Synopsis.

```
boolean   loadFill(
            in short                    PresetNum;
            in SINCGARSFillType         Fill;
            in SINCGARSFillInfoType     FillInfo;
  )
```

4.2.1.2 Parameters.

PresetNum

is the preset number to be filled. Values are 0 to 7 where 0= manual and 7= CUE frequencies.

Fill

designates FillInfo to be hopset, lockout, or TRANSEC data.

```
enum SINCGARSFillType{
        Hopset;
        Lockout;
        TRANSEC;
}
```

FillInfo

is a single fill cell containing hopset, lockout, or TRANSEC fill data.

```
typedef        SINGARSFillInfoType        FillCellType;
```

```
struct   FillCellType{
            boolean                 HopsetLockoutFlag;
            unsigned short          ID;
            unsigned short          FinalSequenceNumber;
            unsigned short          CurrentSequenceNumber;
            octet                   Data1;
            octet                   Data2;
            octet                   Data3;
            octet                   Data4;
            octet                   Data5;
            octet                   Data6;
            octet                   Data7;
            octet                   Data8;
         }
```

4.2.1.3 State.

This service is available in any state except when *zeroize* is in progress.

4.2.1.4 New State.

SINCGARSFillInfoType is loaded (e.g., Hopset/Lockout) into designated preset.

4.2.1.5 Response.

Boolean returns True if the operation was completed successfully; otherwise return is False.

4.2.1.6 Originator.

Service User.

4.2.1.7 Errors/Exceptions.

TBD.

4.2.2 readFillID Service.

readFillID"provides the identity (not the actual Fill data) of a fill element.

4.2.2.1 Synopsis.

```
boolean   readFillID{
             in short                    PresetNum;
             in SINCGARSFillType         Fill;
             out SINCGARS FillIDType     FillID ;
}
```

4.2.2.2 Parameters.

PresetNum

is the selected preset number of the requested fill ID.

Fill

selects hopset, lockout, or TRANSEC information for the read.

```
enum SINCGARSFillType{
        Hopset;
        Lockout;
        TRANSEC;
}
```

FillID

is the fill identifier and fill number in the the selected preset.

```
struct  SINCGARSFillIDType{
        string              SINCGARSFillIdentifier;
        unsigned long       SINCGARSFillIDNo;
}
```

4.2.2.3 State.

Any state.

4.2.2.4 New State.

State remains unchanged

4.2.2.5 Response.

The SINCGARS Fill Identifier string contains the fill data's identifier (e.g. TEK1).

4.2.2.6 Originator.

Service User.

4.2.2.7 Errors/Exceptions.

TBD.

4.2.3 loadSeed Service.

loadSeed provides the ability to transfer base Time of Day(TOD) into the MAC Layer.

4.2.3.1 Synopsis.

```
boolean   loadSeed(
        in short                    SeedNum;
        in SINCGARSSeedType         Seed;
```

```
            in SINCGARSSeedInfoType        SeedInfo;
)
```

4.2.3.2 Parameters.

SeedNum

is ignored by this method.

Seed

is used to distinguish between TOD and WOD. SINCGARS only supports TOD.

```
enum SINCGARSSeedType{
            TOD
}
```

SeedInfo

contains either the TOD or WOD info. SINCGARS only supports TOD.

```
union SINCGARS SeedInfoType{
        TODSeedInfoType          TODSeedInfo;
}
        struct TODSeedInfoType{
            unsigned short          Year;
            unsigned short          Month;
            unsigned short          Day;
            unsigned short          Hour;
            unsigned short          Minutes;
            unsigned short          Seconds;
            unsigned long           Nanoseconds;
        }
```

4.2.3.3 State.

Any state except zeroize.

4.2.3.4 New State.

New base TOD loaded. All net delta TODs are zeroed.

4.2.3.5 Response.

Boolean returns True if the operation was completed successfully; otherwise return is False.

4.2.3.6 Originator.

Service User.

4.2.3.7 Errors/Exceptions.

TBD.

4.2.4 readSeed Service.

readSeed provides the ability to read net Time of Day(TOD) in the selected preset from the MAC Layer.

4.2.4.1 Synopsis.

```
boolean   readSeed(
              in short                      SeedNum;
              in SINCGARSSeedType           Seed;
              out SINCGARSSeedInfoType      SeedInfo;
)
```

4.2.4.2 Parameters.

SeedNum

is used to designate which preset to read net TOD.

Seed

is used to distinguish between TOD and WOD. SINCGARS only supports TOD.

```
enum SINCGARSSeedType{
        TOD
}
```

SeedInfo

contains the TOD info. (SINCGARS only supports TOD.)

```
union SINCGARS SeedInfoType{
        TODSeedInfoType          TODSeedInfo;
        }
        struct  TODSeedInfoType{
                unsigned short          Year;
                unsigned short          Month;
                unsigned short          Day;
                unsigned short          Hour;
                unsigned short          Minutes;
                unsigned short          Seconds;
                unsigned long           Nanoseconds;
        }
```

4.2.4.3 State.

Any state except zeroize.

4.2.4.4 New State.

No change.

4.2.4.5 Response.

Boolean returns True if the operation was completed successfully, otherwise return is False.

4.2.4.6 Originator.

Service User.

4.2.4.7 Errors/Exceptions.

TBD.

4.2.5 zeroize Service.

zeroize provides a method of removing all Fill and Seed data from memory space owned by the TRANSEC function.

4.2.5.1 Synopsis.

boolean zeroize ( )

4.2.5.2 Parameters.

None.

4.2.5.3 State.

Any.

4.2.5.4 New State.

Fill (PresetNum = 0 to 7) is empty and Seed (PresetNum = 0 to 7) is empty.

4.2.5.5 Response.

Boolean returns True when zeroization is successful; otherwise it returns False.

4.2.5.6 Originator.

Service User.

4.2.5.7 Errors/Exceptions.

TBD.

4.3 CHANNEL ERROR CONTROL SERVICE PRIMITIVES.

4.3.1 setChannelErrorControl Service

setChannelErrorControl service is not supported in SINCGARS.

4.4 CHANNEL ACCESS SERVICE PRIMITIVES.

4.4.1 setAccessParameters Service.

setAccessParameters provides the ability to configure SINCGARS with the CSMA parameters index, limit, and offset.

4.4.1.1 Synopsis.

```
boolean setAccessParameters {
        in SINCGARSChannelAccessParameterListType          Parameters
}
```

4.4.1.2 Parameters.

Parameters

is a set of parameters defining one precedence in the CSMA protocol. SINCGARSChannelAccessParameterListType is one CSMA precedence comprised of (1) index which defines the message precedence of this window, (2) limit which assigns the number of a slots in this precedence window, and (3) offset which defines the beginning of this precedence window in each contention interval.

```
struct  SINCGARS ChannelAccessParameterListType{
        short           Index;
        short           Limit;
        short           Offset;
}
```

4.4.1.3 State.

Any state.

4.4.1.4 New State.

New CSMA slot parameters configured and used immediately.

4.4.1.5 Response.

Boolean returns True if the access Parameters are valid ($0 \leq$ index $\leq$TBD, $0 \leq$ limit $\leq$TBD, $0 \leq$offset$\leq$TBD); otherwise it returns False.

4.4.1.6 Originator.

Service User.

4.4.1.7 Errors/Exceptions.

TBD.

4.5 MAC ADDRESS SERVICE.

4.5.1 Bind Address Service.

Provides Service Users with the ability to bind "own address" to the MAC Layer. The MAC address is part of the 188-220 protocol located on the red side of the radio.

4.5.1.1 Synopsis.

The following sequence of name-value pairs are set and read using the *config/query* operations inherited from *Resource* in Core Framework.

Name value pairs are specified in the format name: type: range: .

Name:	Type:	Range:
MACAddress	unsigned long	ownMACAddress

4.5.1.2 State.

Valid in all states.

4.5.1.3 New State.

Own MAC address loaded.

4.5.1.4 Response.

None. Must use Query to check that it was loaded correctly.

4.5.1.5 Originator.

Service User.

4.5.1.6 Errors/Exceptions.

TBD.

4.6 DROP CAPTURE SERVICE PRIMITIVES.

4.6.1 dropCapture Service

Provides the Service User with the ability to terminate an active reception and transition to the Transmit State.

4.6.1.1 Synopsis.

boolean dropCapture ( )

4.6.1.2 Parameters.

None.

4.6.1.3 State.

Valid only during an active message reception; otherwise ignored.

4.6.1.4 New State.

Transmit.

4.6.1.5 Response.

Boolean returns True if an active reception is terminated, otherwise it returns False.

4.6.1.6 Service Originator

Service User.

4.6.1.7 Errors/Exceptions.

TBD.

4.7 QUALITY OF SERVICE(QOS) PRIMITIVES.

4.7.1 Read QOS Service

Provides the Service User with the ability read the QOS information produced by the MAC Layer. SINCGARS MAC Layer does not provide this non-real time service.

4.8 MAC Real Time Packet Service (inherited).

The MAC Real Time Packet Service is used to pass time-critical control and data to and from the MAC Layer. This service inherits from both the Packet Building Block and the PacketSignals Building Block. The error signals are inherited from the Logical Link Control ServiceErrorType. Since control and data are transferred to and from the MAC Layer, both a downstream Service provider interface and an upstream User interface is provided. (Downstream is transferring data toward the antenna while upstream is transferring data away from the antenna.)

4.8.1 getMaxPayloadSize

getMaxPayloadSize returns the maximum number of octets in the data part of the Packet. This is a read-only attribute set at MAC instantiation.

4.8.1.1 Synopsis.

```
readonly  void  getMaxPayloadSize {
                out unsigned short          maxPayloadSize
}
```

4.8.1.2 Parameters.

maxPayloadSize

is the maximum number of data octets allowed in a single SINCGARS MAC pushPacket. This does not include information in the control field of the pushPacket. Note that the upstream and downstream values can be different.

4.8.1.3 State.

Can be read anytime. The value is set at MAC waveform instantiation and can not change.

4.8.1.4 New State.

Fixes maximum data payload size.

4.8.1.5 Response.

None.

4.8.1.6 Originator.

None.

4.8.1.7 Errors/Exceptions.

None.

4.8.2 getMinPayloadSize.

getMinPayloadSize returns the minimum number of octets permitted in the data part of the pushPacket. This is a read only attribute set at MAC instantiation.

4.8.2.1 Synopsis.

```
readonly  void  getMinPayloadSize (
                out unsigned short          minPayloadSize
)
```

4.8.2.2 Parameters.

minPayloadSize

is the minimum number of data octets allowed in a single SINCGARS MAC pushPacket. This does not include information in the control field of the same pushPacket. This value is **TBD** in SINCGARS for both upstream and downstream payloads.

4.8.2.3 State.

Can be read anytime. The value is set at MAC waveform instantiation and can not change.

4.8.2.4 New State.

Fixes minimum data payload size.

4.8.2.5 Response.

None.

4.8.2.6 Originator.

None.

4.8.2.7 Errors/Exceptions.

None.

4.8.3 SINCGARSMACDownStreamProvider Service.

SINCGARSMACDownStreamProvider inherits from SINCGARSDownStreamProviderQueue, Packet (from Packet API), PacketSignals (from Packet API), and PacketErrorType (from LLC API). This down stream service provides the ability to transfer real time data and real time control information into the MAC Layer from the layers above.

4.8.4 SINCGARS MAC Down Stream Provider Queue Service.

4.8.4.1 pushPacket Service.

pushPacket is the method to move real time data and control information into the MAC Layer.

4.8.4.1.1 Synopsis.

```
void    pushPacket (
                in octet                                        priority;
                in MACDownStreamControlType                     control;
                in SINCGARSAPI::CommonTypes::OctetSequence      payload;
)
```

4.8.4.1.2 Parameters.

priority

determines which priority queue the packet is destined for. (0 is lowest priority.)

control

is used to concatenate several data packets into one data stream as required.

```
struct  MACDownStreamControlType{
        short                   attribute1;
        streamControlType       streamControl;
{
```

attribute1

is **TBD.**

```
struct  streamControlType{
        boolean         endOfStream;
        unsigned short  streamID;
        octet           sequenceNumber;
}
```

endOfStream
indicates that this packet is the last packet in the stream.

streamID
is the same for all packets in the same stream. It is different for different streams.

sequenceNumber
defines where the current packet fits in the stream. The first packet has sequence number zero.

4.8.4.1.3 State.

Valid when queue is at low water mark or empty when flowControlSignals are enabled.

4.8.4.1.4 New State.

New data available at the queue.

4.8.4.1.5 Response.

If flowControlSignals are enabled and the queue is now above the high water mark, a high water mark signal will be returned.

4.8.4.1.6 Originator.

Downstream, it is the Service User.

4.8.4.1.7 Errors/Exceptions.

TBD.

4.8.4.2 spaceAvailable Service.

spaceAvailable determines the space available in octets in the specified priority queue.

4.8.4.2.1 Synopsis.

```
unsigned short    spaceAvailable (
                        in octet        priorityQueueID;
)
```

4.8.4.2.2 Parameters

priorityQueueID

specifies which priority queue is being interrogated.

4.8.4.2.3 State.

Valid in all states.

4.8.4.2.4 New State.

No state change.

4.8.4.2.5 Response

The space available in the specified queue in octets.

4.8.4.2.6 Originator.

Downstream Service User.

4.8.4.2.7 Errors/Exceptions.

TBD.

4.8.4.3 enableFlowControlSignals Service.

enableFlowControlSignals enables the sending of high water mark and low water mark signals back to the Service User for queues of all priorities.

4.8.4.3.1 Synopsis.

```
void  enableFlowControlSignals {
            in boolean          enable;
}
```

4.8.4.3.2 Parameters.

enable

permits high water mark and low water mark signals to be sent back to the downstream Service User for all priority queues.

4.8.4.3.3 State.

Valid in all states.

4.8.4.3.4 New State.

Water mark signals either enabled or disabled.

4.8.4.3.5 Response.

None

4.8.4.3.6 Originator.

Downstream Service User.

4.8.4.3.7 Errors/Exceptions.

TBD.

4.8.4.4 enableEmptySignalService.

enableEmptylSignal enables sending an empty signal back to the Service User when all priority queues are empty.

4.8.4.4.1 Synopsis.

```
void  enableEmptySignal {
            in boolean          enable;
}
```

4.8.4.4.2 Parameters.

enable,

when asserted, permits the empty signal to be sent back to the downstream Service User when all priority queues are empty.

4.8.4.4.3 State.

Valid when all priority queues are empty and empty signal is enabled.

4.8.4.4.4 New State.

No change.

4.8.4.4.5 Response.

An empty signal if all priority queues are empty.

4.8.4.4.6 Originator.

Downstream Service User.

4.8.4.4.7 Errors/Exceptions.

TBD.

4.8.4.5 setNumOfPriorityQueues Service.

setNumOfPriorityQueues specifies the number of queues to be used at the downstream Service Provider.

4.8.4.5.1 Synopsis.

```
 void  setNumOfPriorityQueues {
                in octet        numOfPriorities;
 }
```

4.8.4.5.2 Parameters.

numOfPriorities,

specifies the number of priority queues to be provided at the downstream Service Provider. SINCGARS supports **TBD** queues.

4.8.4.5.3 State.

Valid only in **TBD** Mode(s).

4.8.4.5.4 New State.

New number of queues at DownStream Service Provider.

4.8.4.5.5 Response

None

4.8.4.5.6 Originator.

Downstream Service User.

4.8.4.5.7 Errors/Exceptions.

TBD.

4.8.5 SINCGARSMACUpStreamUserQueue Service.

The Upstream User Queue must be inherited by the upstream Service User to receive real time control and real time data from the MAC Layer. The description of the SINCGARSMACUpStreamUser is similar to the SINCGARSDownStreamProvider where all references to downstream are replaced with upstream.

4.8.5.1 pushPacket Service.

pushPacket is the method to move real time data and control information from the MAC Layer to the Service User.

4.8.5.1.1 Synopsis.

```
void    pushPacket {
            in octet                                        priority;
            in MACUpStreamControlType                       control;
            in SINCGARSAPI::CommonTypes:OctetSequence       payload;
}
```

4.8.5.1.2 Parameters.

priority

determines which priority queue the packet is destined for. (0 is lowest priority.)

control

is used to concatenate several data packets into one data stream as required.

```
struct  MACUpStreamControlType{
        short                   attribute1;
        streamControlType       StreamControl;
```

attribute1

is **TBD.**

```
struct  streamControlType{
        boolean         endOfStream;
        unsigned short  streamID;
        octet           sequenceNumber;
}
```

endOfStream

indicates that this packet is the last packet in the stream.

streamID

is the same for all packets in the same stream. It is different for different streams.

sequenceNumber

defines where the current packet fits in the stream. The first packet has sequence number zero.

4.8.5.1.3 State.

Valid when queue is at low water mark or empty when flowControlSignals are enabled.

4.8.5.1.4 New State.

New data available at the queue.

4.8.5.1.5 Response

If high water mark is enabled and the queue is now above the high water mark, a high water mark signal will be returned.

4.8.5.1.6 Originator.

Upstream, it is the Service Provider.

4.8.5.1.7 Errors/Exceptions.

TBD.

4.8.5.2 spaceAvailable Service.

spaceAvailable determines the space available in octets in the specified priority queue.

4.8.5.2.1 Synopsis.

```
unsigned short    spaceAvailable (
                        in octet        priorityQueueID;
)
```

4.8.5.2.2 Parameters

priorityQueueID

specifies which priority queue is being interrogated.

4.8.5.2.3 State.

Valid in all states

4.8.5.2.4 New State.

No state change.

4.8.5.2.5 Response.

The space available in the specified queue in octets.

4.8.5.2.6 Originator.

Upstream, it is the MAC Service Provider.

4.8.5.2.7 Errors/Exceptions.

TBD.

4.8.5.3 enableFlowControlSignals Service.

enableFlowControlSignals enables the sending of high water mark and low water mark signals back to the Service Provider for upstream queues of all priorities.

4.8.5.3.1 Synopsis.

```
void  enableFlowControlSignals (
            in boolean          enable;
)
```

4.8.5.3.2 Parameters.

enable

permits high watermark and low watermark signals to be sent back to the upstream Service Provider from the upstream User Queue for all priority queues.

4.8.5.3.3 State.

Valid in all states

4.8.5.3.4 New State.

Watermark signals either enabled or disabled.

4.8.5.3.5 Response.

None

4.8.5.3.6 Originator.

Upstream, it is the MAC Service Provider.

4.8.5.3.7 Errors/Exceptions.

TBD.

4.8.5.4 enableEmptySignal Service.

enableEmptylSignal enables sending an empty signal back to the MAC Service Provider when all upstream priority queues are empty.

4.8.5.4.1 Synopsis.

```
void  enableEmptySignal {
            in boolean          enable;
}
```

4.8.5.4.2 Parameters.

enable,

when asserted, permits the empty signal to be sent back to the upstream Service Provider when all upstream priority queues are empty.

4.8.5.4.3 State.

Valid when all priority queues are empty and empty signal is enabled.

4.8.5.4.4 New State.

No change

4.8.5.4.5 Response.

An empty signal if all priority queues are empty.

4.8.5.4.6 Originator.

Upstream, it is the MAC Service Provider.

4.8.5.4.7 Errors/Exceptions.

TBD.

4.8.5.5 setNumOfPriorityQueues Service.

setNumOfPriorityQueues specifies the number of queues to be used at the upstream Service User.

4.8.5.5.1 Synopsis.

```
 void  setNumOfPriorityQueues (
                in octet        numOfPriorities;
)
```

4.8.5.5.2 Parameters.

numOfPriorities,

specifies the number of priority queues to be provided at the upstream Service User. SINCGARS supports **TBD** queues.

4.8.5.5.3 State.

Valid only in **TBD** Mode(s).

4.8.5.5.4 New State.

New number of queues at MAC Service User.

4.8.5.5.5 Response.

None.

4.8.5.5.6 Originator.

Upstream, it is the Service Provider.

4.8.5.5.7 Errors/Exceptions.

TBD.

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.

Intentionally left blank.

6 UTILIZATION OF SCA BUILDING BLOCKS.

This API makes use of MAC and Generic Packet Building Blocks.

7 PRECEDENCE OF SERVICE PRIMITIVES.

There is no precedence of primitives.

8 SERVICE USER GUIDELINES.

No guidelines for implementing a Service User that are independent of the implementation of the Service Provider have been identified.

9 SERVICE PROVIDER-SPECIFIC INFORMATION.

No Service Provider-Specific information has been identified.

10 IDL.

10.1 COMMON INTERFACES.

```
//Source file: H:/JTRS/SYSJTRS/api/rose
models/ITTBBIDL/CommonInterfacesModules.idl

#ifndef __COMMONINTERFACESMODULES_DEFINED
#define __COMMONINTERFACESMODULES_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

module SINCGARSAPI {

      module CommonInterfaces {

            interface ChannelErrorControl {
                  /*
                  @roseuid 39EB177F0102 */
                  void channelErrorControl (
                        in boolean ErrorControl
                        );

            };

            interface DropCapture {
                  /*
                  @roseuid 39C643F203D4 */
                  boolean dropCapture ();

            };

            interface ReceiveTermination {
                  /*
                  @roseuid 39D0073C0171 */
                  boolean dropCapture ();

                  /*
                  @roseuid 39D0ADCC0174 */
                  boolean abortReceive ();

            };

            interface PhysicalManagement {
                  attribute unsigned short minTU;
                  attribute unsigned short maxTU;
            };

            interface TransmitInhibit {
                  /*
                  @roseuid 39D0B62B0021 */
                  boolean inhibitTransmit (
                        in boolean Inhibit
                        );
```

```
            };

            interface PacketSignals {
                  /* This operation is a call event back to the PacketAPI
client indicating that a queue has reach the high watermark.  If priority or
multiple queues are being supported then the priorityQueueID indicates which
queue has reached the high watermark.
                  @roseuid 38F3442F01B8 */
                  oneway void signalHighWatermark (
                        in octet priorityQueueID
                        );

                  /* This operation is a call event back to the PacketAPI
client indicating that the queue has reach the low watermark.   If priority
or multiple queues are being supported then this indicates that the sum total
of all the queues has reached the low watermark.
                  @roseuid 38F3446F025A */
                  oneway void signalLowWaterMark (
                        in octet priorityQueueID
                        );

                  /* This operation is a call event back to the PacketAPI
client indicating that the queue has emptied.   If priority or multiple
queues are being supported then this indicates that the sum total of all the
queues has reached zero.
                  @roseuid 38FE26CF02FA */
                  oneway void signalEmpty ();

            };

            interface AudioControl {
                  attribute boolean SidetoneEnabled;
                  attribute short MicrophoneGainIndB;
                  attribute boolean AudioOutputEnabled;
                  attribute short OutputGainIndB;

                  /*
                  @roseuid 39ECA7A50255 */
                  void enableRTSAndCTS (
                        in boolean Enable
                        );

                  /*
                  @roseuid 39ECA7C601B2 */
                  void setCTS (
                        in boolean CTS
                        );

            };

            interface AudiodeviceSignals {
                  /*
                  @roseuid 39E4A192016E */
                  void SignalRTS (
                        in boolean RTS
                        );
```

```
            };
      };
};
#endif
```

10.2 COMMON TYPES.

```
//Source file: H:/JTRS/SYSJTRS/api/rose
models/ITTBBIDL/CommonTypesModules.idl

#ifndef __COMMONTYPESMODULES_DEFINED
#define __COMMONTYPESMODULES_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

module SINCGARSAPI {

      module CommonTypes {

            enum ServiceErrorType {
                  ERROR_BAD_SAP,
                  ERROR_BAD_ADDRESS,
                  ERROR_NO_ACCESS,
                  ERROR_INVALID_STATE,
                  ERROR_BAD_CORRELATION,
                  ERROR_BAD_DATA,
                  ERROR_UNSUPPORTED,
                  ERROR_NOT_ENABLED,
                  ERROR_TOO_MANY,
                  ERROR_BOUND,
                  ERROR_NO_AUTO,
                  ERROR_NO_XIDAUTO,
                  ERROR_NO_TESTAUTO,
                  ERROR_NO_ADDRESS,
                  ERROR_BAD_QOS_PARAMETERS,
                  ERROR_UNDELIVERABLE
            };

            /* Identify this data stream for acknowledgement processing,
cancelation of transmission,etc.  */

            struct StreamControlType {
                  unsigned short streamID;
                  /* Indicates that the last symbol of this hop is an end of
stream.  */
                  boolean endOfStream;
                  /* Sequence number of the hop within the stream sequence.
The waveform application sets this value to zero at every occurrence of a
start of stream.  If value is set to zero it indicates beginning of stream.
*/
                  octet sequenceNum;
            };

            struct TimeType {
                  unsigned long seconds;
                  unsigned long nanoSec;
            };

            struct BeepType {
```

```
                  short BeepLevelIndB;
                  unsigned short DurationInMs;
                  unsigned short FrequencyInHz;
            };

            typedef unsigned short IdType;

            struct DescriminatorType {
                  unsigned short DataTypeDescriminator;
                  unsigned short BeepTypeDescriminator;
                  unsigned short AlarmTypeDescriminator;
                  unsigned short SeedInfoDescriminator;
            };

            enum ErrType {
                  PktUsageErr,
                  PktErrNo
            };

            union PacketErrorType switch(ErrType) {
                  case PktUsageErr:
SINCGARSAPI::CommonTypes::ServiceErrorType usageError;
                  case PktErrNo: unsigned long errNo;
            };

            typedef sequence<octet> OctetSequence;

      };

};

#endif
```

10.3 NON-REAL TIME.

```
//Source file: H:/JTRS/SYSJTRS/api/rose models/ITTBBIDL/NRTMAC.idl

#ifndef __NRTMAC_DEFINED
#define __NRTMAC_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

#include "CommonInterfacesModules.idl"

module SINCGARSAPI {

      module NRTMAC {

            module MACCommonUtility {

                  enum SINCGARSPowerModeType {
                        Hi,
                        LO,
                        MED,
                        PA
                  };

                  interface SINCGARSMACCommonUtility {
                        attribute unsigned long maxTU;
                        attribute unsigned long minTU;

                        /*
                        @roseuid 39E2243703D8 */
                        boolean activateChannel (
                              in short PresetNum
                              );

                        /*
                        @roseuid 39E22438000D */
                        boolean setRFPowerControl (
                              in SINCGARSPowerModeType PowerMode
                              );

                        /*
                        @roseuid 39E2263D0179 */
                        void getRFPowerControl (
                              out SINCGARSPowerModeType PowerMode
                              );

                        /*
                        @roseuid 39E224380018 */
                        void getMinTU (
                              out unsigned long MinTU
                              );

                        /*
                        @roseuid 39E224380021 */
```

```
            void getMaxTU (
                  out unsigned long MaxTU
                  );

            /*
            @roseuid 39E4ADA9030F */
            void startScan ();

            /*
            @roseuid 39E4ADD100E5 */
            void stopScan ();

            /*
            @roseuid 39E4ADE80138 */
            void clearPreset (
                  in short PresetNum
                  );

            /*
            @roseuid 39E4AE0A01F6 */
            void clearAllPresets ();

      };

};

module TRANSEC {

      struct SINCGARSFillIDType {
            string SINCGARSFillIdentifier;
            unsigned long SINCGARSFillIDNo;
      };

      enum SINCGARSFillType {
            Hopset,
            Lockout,
            TRANSEC
      };

      enum SINCGARSSeedType {
            TOD
      };

      struct TODSeedInfoType {
            unsigned short Year;
            unsigned short Month;
            unsigned short Day;
            unsigned short Hour;
            unsigned short Minutes;
            unsigned short Seconds;
            unsigned long NanoSeconds;
      };

      struct FillCellType {
            boolean HopsetLockoutFlag;
            unsigned short ID;
            unsigned short FinalSequenceNumber;
```

```
        unsigned short CurrentSequeceNumber;
        octet Data1;
        octet Data2;
        octet Data3;
        octet Data4;
        octet Data5;
        octet Data6;
        octet Data7;
        octet Data8;
};

typedef sequence<FillCellType> SINCGARSFillInfoType;

enum SeedInfoDiscriminator {
        TODSeed
};

union SINCGARSSeedInfoType switch(SeedInfoDiscriminator) {
        case TODSeed: TODSeedInfoType TODSeedInfo;
};

interface SINCGARSTRANSEC {
        attribute short maxFILLLength;
        attribute short maxSEEDLength;
        attribute boolean zeroized;

        /*
        @roseuid 39EB5682014D */
        boolean loadFill (
                in short PresetNum,
                in SINCGARSFillType Fill,
                in SINCGARSFillInfoType FillInfo
                );

        /*
        @roseuid 39EB56820159 */
        boolean readFillID (
                in short PresetNum,
                in SINCGARSFillType Fill,
                out SINCGARSFillIDType FillID
                );

        /*
        @roseuid 39EB5682016B */
        boolean loadSeed (
                in short SeedNum,
                in SINCGARSSeedType Seed,
                in SINCGARSSeedInfoType SeedInfo
                );

        /*
        @roseuid 39EB56820176 */
        boolean readSeed (
                in short SeedNum,
                in SINCGARSSeedType Seed,
                out SINCGARSSeedInfoType SeedInfo
                );
```

```
                        /*
                        @roseuid 39EB56820181 */
                        boolean zeroize ();

                  };

            };

            module ChannelAccess {

                  struct SINCGARSChannelAccessParameterType {
                        short Index;
                        short Limit;
                        short Offset;
                  };

                  typedef sequence<SINCGARSChannelAccessParameterType>
SINCGARSChannelAccessParameterListType;

                  interface SINCGARSChannelAccess {
                        /*
                        @roseuid 39E2360C03BD */
                        boolean setAccessParameters (
                              in SINCGARSChannelAccessParameterListType
Parameters
                              );

                        /*
                        @roseuid 39E237F500C7 */
                        void setCSMAIGAP (
                              in short IGAP
                              );

                        /*
                        @roseuid 39F99B79030C */
                        void setCSMAPrecedence (
                              in short Precedence
                              );

                  };

            };

            module ChannelErrorControl {

                  interface SINCGARSChannelErrorControl :
CommonInterfaces::ChannelErrorControl {
                  };

            };

            module MACAddressing {

                  interface SINCGARSAddressType {
                        attribute unsigned long Address;
                  };
```

```
                  interface SINCGARSMACAddressing {
                        /*
                        @roseuid 39E2FB84025E */
                        boolean bindAddress (
                              in SINCGARSAddressType address
                              );

                  };

            };

            module DropCapture {

                  interface SINCGARSDropCapture :
CommonInterfaces::DropCapture {
                  };

            };

      };

};

#endif
```

10.4 REAL-TIME.

```
//Source file: H:/JTRS/SYSJTRS/api/rose models/ITTBBIDL/RTMAC.idl

#ifndef __RTMAC_DEFINED
#define __RTMAC_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

#include "CommonInterfacesModules.idl"
#include "CommonTypesModules.idl"

module SINCGARSAPI {

      module RTMAC {

            struct MACDownStreamControlType {
                  short attribute1;
            };

            interface SINCGARSMACDownStreamProviderQueue {
                  /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */

                  attribute unsigned short maxPayloadSize;
                  attribute unsigned short minPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
                  @roseuid 39E30C2100B5 */
                  void pushPacket (
                        in octet priority,
                        in MACDownStreamControlType control,
                        in SINCGARSAPI::CommonTypes::OctetSequence payload
                        );

                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39E30C2100C0 */
                  unsigned short spaceAvailable (
                        in octet priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39E30C2100C9 */
                  void enableFlowControlSignals (
                        in boolean enable
                        );

                  /* This operation allows the client to turn theEmpty Signal
ON and OFF.
                  @roseuid 39E30C2100CB */
```

```
                void enableEmptySignal (
                      in boolean enable
                      );

                /*
                @roseuid 39E30C2100D4 */
                void setNumOfPriorityQueues (
                      in octet numOfPriorities
                      );

            };

            interface SINCGARSMACDownStreamProvider :
SINCGARSMACDownStreamProviderQueue, CommonInterfaces::PacketSignals {
            };

            struct MACUpStreamControlType {
                  short attribute1;
            };

            interface SINCGARSMACUpStreamUserQueue {
                  /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */

                  attribute unsigned short maxPayloadSize;
                  attribute unsigned short minPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
                  @roseuid 39E2FE6C0309 */
                  void pushPacket (
                        in octet priority,
                        in MACUpStreamControlType control,
                        in SINCGARSAPI::CommonTypes::OctetSequence payload
                        );

                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39E2FE6C031C */
                  unsigned short spaceAvailable (
                        in octet priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39E2FE6C031E */
                  void enableFlowControlSignals (
                        in boolean enable
                        );

                  /* This operation allows the client to turn theEmpty Signal
ON and OFF.
                  @roseuid 39E2FE6C0327 */
                  void enableEmptySignal (
                        in boolean enable
                        );
```

```
                  /*
                  @roseuid 39E2FE6C0332 */
                  void setNumOfPriorityQueues (
                        in octet numOfPriorities
                        );

            };

            interface SINCGARSMACUpStreamUser : SINCGARSMACUpStreamUserQueue,
CommonInterfaces::PacketSignals {
                  /*
                  @roseuid 39E4CCD9039B */
                  void signalError (
                        in SINCGARSAPI::CommonTypes::PacketErrorType error
                        );

            };

      };

};

#endif
```

11 UML.

UML class and component diagrams for the Service Definition are included in Section 3.