UNCLASSIFIED

AD NUMBER

ADB120259

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; MAR 1988. Other requests shall be referred to Air Force Armament Lab., Eglin AFB, FL. This document contains export-controlled technical data.

AUTHORITY

AFSC/MNOL Wright Lab ltr dtd 13 Feb 1992

THIS PAGE IS UNCLASSIFIED

DTTC FILE COPY

UNCLASSIFIED

_	RE	PORT D	OCUMENTATIO	N PAGE			Form OMB	Approved No. 0704-018
1a, REPORT SI	CURITY CLASSIFICATION			15 RESTRICTIVE	MARKINGS			
Unclas	sified							
2a. SECURITY	CLASSIFICATION AUTHOR	RITY		3. DISTRIBUTION	AVAILABILITY	OF REPOR	RT	
2b DECLASSIE		G SCHEDUIL	<u> </u>	Distributi	ion author	ized to	U.S. 0	overnme
			5	Agencies	and their	contra	ctors;C	(over)
4. PERFORMIN	G ORGANIZATION REPOR	RT NUMBER	(\$)	5. MONITORING	ORGANIZATION	REPORT	NUMBER(S)	_
				AFATL-T	R-88-18,	VOI 12		
6a. NAME OF	PERFORMING ORGANIZA		6b. OFFICE SYMBOL	Za. NAME OF M	ONITORING OR	GANIZATIC	N	
McDon	nell Douglas		(If applicable)	Acromoch	anice Divi	sion		
Astron	autics Company			Aeromeen	antes Divi	51011		
6c. ADDRESS (City, State, and ZIP Code,	9		76. ADDRESS (Ch	ty, State, and Z	IP Code)		
P.O. H	30x 516			Air Force	e Armamen B. FL. 325	42-5434	l	
St. Lo	uis, MO 03100			Lgun Ar	5, 11 020	10 010		
8a. NAME OF	FUNDING / SPONSORING	I	86. OFFICE SYMBOL	9. PROCUREMEN	T INSTRUMENT	IDENTIFIC	ATION NUM	ABER
ORGANIZA STARS	NON S Joint Program (Office	(If applicable)	F08635-8	6-C-0025			
STARC	City State and Zin Codel					EPC		
Room	SD139 (1211 Fern	i St)		PROGRAM	PROJECT	TASK		WORK UNIT
The Pe	entagon	0001		ELEMENT NO.	NO.	NO.		ACCESSION I
Washin	igton DC 20301-	3081		63756D	921C	GZ		57
11. TITLE (Incl	ude Security Classification	n)				•• •		
Commo	n Ada Missile Pa	ickage (CAMP) Project:	: Missile Soft	ware Parts	, Vol	[2:	
Detail	AUTHOR(S)	ts (vol	(-12)			···		_
D Mc	Nicholl, S. Coher	n. C. P:	almer, et al.					
13a. TYPE OF	REPORT 13	b. TIME CO	VERED Man 89	14. DATE OF REPO	RT (Year, Mon	th, Day)	15. PAGE	OUNT
Techn	ical_NoteF	ROM	p 00 10 Mai 00	March 198	38		230	
16. SUPPLEME	NTARY NOTATION	SUBJ	ECT TO EXPOR	T CONTROL	LAWS.			
	Availabili	ty of th	is report is spe	orified on ver	rso of fron	t cover	•	(ov
ī	11 / 0110.511			ectified off act				
17.	COSATI CODES		18. SUBJECT TERMS	Continue on reven	e if necestary a	nd identii	y by block	number)
17. FIELD	COSATI CODES GROUP SUB-GI	ROUP	18. SUBJECT TERMS (Reusable S Ada, Parts	Continue on reven Oftware, Miss Composition	sile Softwa Svstems	re, Soit	y by block itware are Par	number) Generato: ts
17. FIELD	COSATI CODES GROUP SUB-GI	ROUP	18. SUBJECT TERMS (Reusable S Ada, Parts	Continue on reven oftware, Miss Composition	sile Softwa Systems	nd identii re, Soi , Softw	y by block itware are Par	<i>number)</i> Generato: ts
17. FIELD	COSATI CODES GROUP SUB-GI	necessary a	18. SUBJECT TERMS (Reusable S Ada, Parts	Continue on revers oftware, Miss Composition	e if necessary a sile Softwa Systems	nd identif re, Soi , Softw	y by block itware (are Par	number) Generator ts
17. FIELD 19. ABSTRACT The ol	COSATI CODES GROUP SUB-GI (Continue on reverse if pjective of the CA	ROUP	18. SUBJECT TERMS (Reusable S Ada, Parts nd identify by block n ogram is to dem	Continue on reverse oftware, Miss Composition	feasibility	of reu	y by block itware (are Par sable A	number) Generato: ts da softw
17. FIELD 19. ABSTRACT The ol parts	COSATI CODES GROUP SUB-GI (Continue on reverse if ojective of the CA in a real-time em f miscilo flight co	necessary a AMP pro ibedded	is. SUBJECT TERMS (Reusable S Ada, Parts ogram is to dem application are	Continue on reverse oftware, Miss Composition Composition onstrate the a; the domai required the	feasibility feasibility	of reu of the	y by block ftware (are Par sable A demons	da softw tration w
17. FIELD 19. ABSTRACT The ol parts that o within	COSATI CODES GROUP SUB-GI (Continue on reverse if ojective of the CA in a real-time em f missile flight so that domain be y	necessary a AMP pro ibedded oftware verified	18. SUBJECT TERMS (Reusable S Ada, Parts ogram is to dem application are systems. This (in order to ju	Continue on reverse oftware, Miss Composition composition onstrate the a; the domais required that astify the dev	feasibility feasibility n chosen f at the exis	of reu of reu of the tence of	y by block ftware (are Par sable A demons f comm s for th	da softw tration wonality at domai
17. FIELD 19. ABSTRACT The ol parts that o within and th	COSATI CODES GROUP SUB-GI (Continue on reverse if ojective of the Ca in a real-time em f missile flight so that domain be to at software parts	AMP pro bedded oftware verified s be des	18. SUBJECT TERMS (Reusable S Ada, Parts ogram is to dem application are systems. This (in order to ju signed which ac	Continue on reverse oftware, Miss Composition constrate the a; the domais required the ustify the devi	feasibility feasibility n chosen f at the exis velopment areas ider	of reu of reu of reu or the tence o of part	y by block tware far are Par sable A demons of commu- s for th An as	da softw tration w onality at domai sociated
17. FIELD 19. ABSTRACT The ol parts that or within and th parts	COSATI CODES GROUP SUB-GI (Continue on reverse if ojective of the CA in a real-time em f missile flight so that domain be v lat software parts system was devel	necessary a AMP pro- ibedded oftware verified s be des loped to	18. SUBJECT TERMS (Reusable S Ada, Parts ogram is to dem application are systems. This (in order to justice of the support parts)	Continue on reverse oftware, Miss Composition Composition onstrate the a; the domai required that istify the dev ddress those usage. Volt	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th	of reu of reu of reu or the tence of part ntified.	y by block ftware (are Par sable A demons f comm s for th An as ument i	da softw tration w onality at domai sociated s the Use
17. FIELD 19. ABSTRACT The ol parts that o within and th parts Guide	COSATI CODES GROUP SUB-GI (Continue on reverse if ojective of the CA in a real-time em f missile flight so that domain be v iat software parts system was devel to the CAMP Sof	ROUP necessary a AMP pro ibedded oftware verified is be des loped to ftware p	18. SUBJECT TERMS (Reusable S Ada, Parts ogram is to dem application are systems. This (in order to ju signed which ac support parts arts; Volume 2	Continue on reverse oftware, Miss Composition Composition Constrate the a; the domain required the astify the deviderss those usage. Volton is the Version	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip	of reu of reu of reu or the tence o of part. ntified. nis doct	y by block ftware (are Par sable A demons of common s for th An as ument i	da softw tration w onality at domai sociated s the Use ; Volume
17. FIELD 19. ABSTRACT The ol parts that or within and th parts Guide is the	COSATI CODES GROUP SUB-GI (Continue on reverse if ojective of the CA in a real-time em f missile flight so that domain be that domain be system was devel to the CAMP Sof Software Produc	ROUP necessary a AMP pro- bedded oftware verified is be des loped to ftware p it-Specification the specification the	18. SUBJECT TERMS (Reusable S Ada, Parts Ada, Parts ogram is to dem application are systems. This (in order to ju signed which ac support parts arts; Volume 2 fication; Volume	Continue on reverse offware, Miss Composition Composition onstrate the a; the domain required that astify the dev ddress those usage. Volto is the Version es 4-6 contain Documents	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu or the tence o of part ntified. his doct tion Do Level 1	y by block ttware (are Par sable A demons of common s for th An as ument i ocument Design	da softw tration w onality at domai sociated s the Use ; Volume Documen
17. FIELD 19. ABSTRACT The ol parts that ol within and th parts Guide is the and,	COSATI CODES GROUP SUB-G Continue on reverse if Djective of the Calin a real-time em f missile flight so that domain be that system was devel to the CAMP Sof Software Produc Jolumes 7-12 continues	AMP pro bedded oftware verified is be des loped to ftware p st-Specifi itain the	18. SUBJECT TERMS (Reusable S Ada, Parts ogram is to dem application are systems. This (in order to justice of the support parts arts; Volume 2 ication; Volume	Continue on reven oftware, Miss Composition Composition constrate the astify the devi dress those usage." Volu- is the Versic es 4-6 contain Documents.	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu of reu or the tence o of part. nis doct tion Do Level I	x by block are Par sable A demons of common s for th An as ument i ocument Design	da softw tation w onality at domai sociated s the Use ; Volume Documen
17. FIELD 19. ABSTRACT The ol parts that o within and th parts Guide is the and, V	COSATI CODES GROUP SUB-G Continue on reverse if Djective of the CA in a real-time em f missile flight so that domain be v at software parts system was devel to the CAMP Sof Software Produc Jolumes 7-12 cont	AMP pro bedded oftware verified s be des loped to ftware p st-Specifi tain the patting	18. SUBJECT TERMS (Reusable S Ada, Parts ogram is to dem application are systems. This (in order to justice of the support parts arts; Volume 2 Ication; Volume	Continue on reverse oftware, Miss Composition Composit	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu or the tence o of part. ntified. nis doct tion Do Level 1	x by block are Par sable A demons of common s for th An as ument i boument i Design	da softw tration w onality at domai sociated s the Use ; Volume Documen
17. FIELD 19. ABSTRACT The ol parts that o within and th parts Guide is the and,	COSATI CODES GROUP SUB-G (Continue on reverse if ojective of the Ca in a real-time em f missile flight so that domain be at software parts system was devel to the CAMP Sof Software Produc Volumes 7-12 cont	AMP pro abedded oftware verified s be des loped to ftware p st-Specifi tain, the pact	18. SUBJECT TERMS (Reusable S Ada, Parts Ada, Parts ogram is to dem application are systems. This (in order to ju signed which ac support parts arts; Volume 2 fication; Volume	Continue on reverse oftware, Miss Composition Composition a; the domain required the astify the dev ddress those usage. Volu- is the Version es 4-6 contain Documents.	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu or the tence o of part ntified. his docu tion Do Level I	sable A demons of commons of comm	da softw tration w onality hat domai sociated s the Use ; Volume Documen DECT
17. FIELD 19. ABSTRACT The ol parts that o within and th parts Guide is the and, v	COSATI CODES GROUP SUB-G Continue on reverse if Djective of the CA in a real-time em f missile flight so that domain be that software parts system was deve to the CAMP Sof Software Produc Volumes 7-12 cont	ROUP necessary a AMP pro- bedded oftware verified is be des loped to ftware p oftware p oftware p oftware for hear of the part of the	18. SUBJECT TERMS of Reusable S Ada, Parts Ada, Parts ogram is to dem application are systems. This (in order to jusigned which ac support parts arts; Volume 2 ication; Volume	Continue on reven oftware, Miss Composition Composition onstrate the ea; the domain required that istify the dev ddress those usage. Volu- is the Version es 4-6 contain Documents.	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu of reu or the tence o of part. ntified. nis doct tion Do Level I	sable A demons of common s for th An as ument i ocument Design	da softw tation w onality at domai sociated s the Use ; Volume Document DICC LECT PR 0 7 19
17. FIELD 19. ABSTRACT The ol parts that o within and th parts Guide is the and, v	COSATI CODES GROUP SUB-GI (Continue on reverse if ojective of the CA in a real-time em f missile flight so that domain be that domain be to the CAMP Sof Software Produc Volumes 7-12 continues	ROUP necessary a AMP pro- bedded oftware verified is be des loped to ftware p st-Specifitain, the pact- fact- is BSTRACT	18. SUBJECT TERMS (Reusable S Ada, Parts Ada, Parts ogram is to dem application are systems. This (in order to ju signed which ac support parts arts; Volume 2 lication; Volume	Continue on reverse oftware, Miss Composition Composition onstrate the a; the domain required the stify the dev ddress those usage. Volu- is the Versid es 4-6 contain Documents.	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu or the tence o of part ntified. his doci tion Do Level I	sable A demons of commons of commons of commons s for th An as ument i ocument Design	da softw tration w onality at domai sociated s the Use ; Volume Documen DTIC LECT PR 0 7 %
17. FIELD 19. ABSTRACT The ol parts that o within and th parts Guide is the and, V 20. DISTRIBUT UNCLASS	COSATI CODES GROUP SUB-G Continue on reverse if Djective of the CA in a real-time em f missile flight sc that domain be that domain be system was deve to the CAMP Sof Software Produc Volumes 7-12 cont ION/AVAILABILITY OF A	ROUP necessary a AMP pro- bedded oftware verified is be des eloped to ftware p st-Specification, the part of the is and the part of the is and the start of the is a start of th	 18. SUBJECT TERMS (Reusable S Ada, Parts Ada, Parts ogram is to dem application are systems. This (in order to jusigned which ac support parts arts; Volume 2 fication; Volume Detail Design 	Continue on reverse oftware, Miss Composition Composit	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu or the tence o of part. his doct tion Do Level I	sable A demons f comme s for th An as ument i ocument Design	da softw tration w onality at domai sociated s the Use ; Volume Documen DICC PR 0 7 19
17. FIELD 19. ABSTRACT → The ol parts that o within and th parts Guide is the and, v 20. DISTRIBUT □UNCLASS 22a. NAME O Christ	COSATI CODES GROUP SUB-GI (Continue on reverse if ojective of the CA in a real-time em f missile flight so that domain be tat software parts system was devel to the CAMP Sof Software Produc Volumes 7-12 cont ION/AVAILABILITY OF A IFIED/UNLIMITED S RESPONSIBLE INDIVIDUA	ROUP necessary a AMP pro- bedded oftware verified is be des eloped to ftware p st-Specification train_the pact- pact- issue as RP AL	 IS. SUBJECT TERMS (Reusable S Ada, Parts Ada, Parts Identify by block n ogram is to dem application are systems. This (in order to ju signed which ac support parts arts; Volume 2 dication; Volume Detail Design 	Continue on reverse oftware, Miss Composition composition onstrate the a; the domais required that stify the dev ddress those usage." Volu- is the Versic es 4-6 contais Documents.	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu or the tence o of part ntified. his doci tion Do Level I	sable A demons of comm s for th An as ument i ocument Design	Cenerator Generator ts da softw tration w onality hat domai sociated s the Use ; Volume Document Docum
17. FIELD 19. ABSTRACT The ol parts that o within and th parts Guide is the and, V 20. DISTRIBUT UNCLASS 22a. NAME OF Christ DD Form 147	COSATI CODES GROUP SUB-G Continue on reverse if Djective of the CA in a real-time em f missile flight sc that domain be that domain be that software parts system was deve to the CAMP Sof Software Produc Volumes 7-12 Software Produc Volumes 7-12 Software Software stat Software Produc Volumes 7-12 Software Software Software Produc Volumes 7-12 Software S	ROUP necessary a AMP pro- bedded oftware verified is be des eloped to ftware p ot-Specification tain the pat-fication AMBSTRACT SAME AS RP AL	 18. SUBJECT TERMS (Reusable S Ada, Parts Ada, Parts ogram is to dem application are systems. This (in order to jusigned which ac support parts arts; Volume 2 fication; Volume 2 fication; Volume 2 	Continue on reverse oftware, Miss Composition Composit	feasibility feasibility n chosen f at the exis velopment areas ider ume 1 of th on Descrip n the Top-	of reu of reu or the tence o of part ntified. his doct tion Do Level I	sable A demons of commons of comm	Generato Generato ts da softw tration w onality at domai sociated s the Use ; Volume Document Document Document CLECT PR 0 7 19 FXG F THIS PAGE

UNCLASSIFIED

6....

. 1

ير د

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

this-report documents test and ovulation; distribution limitation applied March 1988. Other requests for this document must be referred to AFATL/FXG, Eglin AFB, Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

These technical notes accompany the CAMP final report AFATL-TR-85-93 (3 Vols)

UNCLASSIFIED

AFATL-TR-88-18, Vol 12

SOPTVARE DETAILED DESIGN DOCUMENT

FOR THE

MISSILE SOFTWARE PARTS

OF THE

COMMON ADA MISSILE PACKAGE (CAMP) PROJECT

CONTRACT F08635-86-C-0025

CDRL SEQUENCE NO. COO7



Acces	sion For
NTIS	GRA&I
DTIC	тав 🕱
Unann	ounced
Justi	fication
By	
Distr	ibution/
Avai	lability Codes
	Avail and/or
Dist	Special
C-2	57 . W

30 OCTOBER 1987

Distribution authorized to U.S. Government agencies and their contractors only; this report desuments test and evaluation; distribution limitation applied July 1987. Other requests for this document must be referred to the Air Force Armament Laboratory (FXG) Eglin Air Force Base, Florida 32542 – 5434.

DESTRUCTION NOTICE – For classified documents, follow the procedures in DoD 5220.22 – M, Industrial Security Manual, Section II – 19 or DoD 5200.1 – R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

<u>WARNING:</u> This document contains technical data whose export is restricted by the Arms Export Control Act (Title 22, U.S.C., Sec. 2751, <u>et seq.</u>) or the Export Admin – istration Act of 1979, as amended (Title 50, U.S.C., App. 2401, <u>et seq.</u>). Violations of these export laws are subject to severe criminal penalties. Disseminate in accordance with the provisions of AFR 80 – 34.

88

4

6

131

AIR FORCE ARMAMENT LABORATORY Air Force Systems Command United States Air Force Eglin Air Force Base, Florida

3.3.7 ABSTRACT MECHANISMS

(23)

¢,

.

Page 1730

.

(This page intentionally left blank.)

٠

3.3.7.1 ABSTRACT DATA STRUCTURES TLCSC P691 (CATALOG #P330-0)

This package contains the bodies of the generic packages required to define and manipulate the following abstract data structures:

o bounded FIFO buffer o unbounded FIFO buffer o nonblocking circular buffer o unbounded priority queue o bounded stack o unbounded stack

It also contains the package required by the unbounded parts to handle the manipulation of their available space lists.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

 Name
 Requirements Allocation

 Bounded_FIF0_Buffer
 R125

 Unbounded_FIF0_Buffer
 R164

 Nonblocking_Circular_Buffer
 R126

 Unbounded_Priority_Queue
 R165

 Bounded_Stack
 R166

 Unbounded_Stack
 R167

3.3.7.1.2 LOCAL ENTITIES DESIGN

Packages:

The following table describes the packages maintained local to this part:

1	Name		Туре		Description	
	Available_Space_ List_Operations		generic package	 	Contains a set of functions to retrieve a node from and add a node to an available space list	

3.3.7.1.3 INPUT/OUTPUT

None.

3.3.7.1.4 LOCAL DATA

None.

Page 1731



3.3.7.1.5 PROCESS CONTROL Not applicable.

3.3.7.1.6 PROCESSING

The following describes the processing performed by this part: package body Abstract_Data_Structures is

-- --separate package bodies

package body Bounded_FIFO_Buffer is separate; package body Unbounded_FIFO_Buffer is separate; package body Nonblocking_Circular_Buffer is separate; package body Unbounded_Priority_Queue is separate; package body Bounded_Stack is separate; package body Unbounded_Stack is separate; package body Unbounded_Stack is separate; package body Available_Space_List_Operations is separate;

end Abstract_Data_Structures;

3.3.7.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.7.1.8 LIMITATIONS

None.

3.3.7.1.9 LLCSC DESIGN

3.3.7.1.9.1 AVAILABLE SPACE LIST OPERATIONS PACKAGE DESIGN

This package contains a set of routines used to manipulate an available space list which is maintained local to the part instantiating this package.

The first routine, New Node, will return a node to the calling routine. If a node is available in the available space list, the node will be retrieved from there. If not, a new node will be dynamically allocated. If no memory is available for the allocation, a STORAGE ERROR exception is raised.

Page 1732

The second routine, Save_Node, places a node in the available space list.

The third routine, Save_List, places a list of nodes in the available space list.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.9.1.1 REQUIREMENTS ALLOCATION

This part helps meet CAMP requirements R164, R165, R167.

3.3.7.1.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

 Name
 Type
 Description

 Nodes
 limited private
 A single element in the available space list

 Pointers
 access Nodes
 A pointer to an element in the available

 space list
 space list

Data objects:

The following table summarizes the generic formal objects required by this part. All of these objects are in/out parameters and are changed by calles to the enclosed routines.

1	Name	Туре	Value	Description	
	Available_ Length	INTEGER	N/A	Length of the available space list	
	Available_Head	Pointers	N/A	Points to the first element in the available space list	ĺ
	Available_Tail	Pointers 	N/A 	Points to the last element in the available space list	İ

Subprograms:

The following table describes the generic formal subprograms required by this part:

Name Type Description
Dot_Next function Given a pointer to a node, this function returns a pointer to the next node in the list Set_Next procedure Given two points, A and B, sets A.Next equal to B
3.3.7.1.9.1.4 LOCAL DATA None.
3.3.7.1.9.1.5 PROCESS CONTROL
Not applicable.
3.3.7.1.9.1.6 PROCESSING
The following describes the processing performed by this part:
<pre>generic type Nodes is limited private; type Pointers is access Nodes; Available Length : in out INTEGER; Available Head : in out Pointers; Available Tail : in out Pointers; with function Dot Next (Ptr : in Pointers) return Pointers is <>; with procedure Set_Next (Ptr : in Pointers; Ptr_dot_Next : in Pointers) is <>; package Available_Space_List_Operations is</pre>
function New_Node return Pointers;
<pre>procedure Save_Node (Saved_Node : in Pointers);</pre>
procedure Save_List (Saved_Head : in Pointers; Saved_Tail : in Pointers; Node_Count : in POSITIVE);
<pre>end Available_Space_List_Operations;</pre>
3.3.7.1.9.1.7 UTILIZATION OF OTHER ELEMENTS
None.

3.3.7.1.9.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Ø

8

6

	Name		When/Why Raised	
	STANDARD.STORAGE_ERROR		Raised during elaboration of this package if an attempt is made to allocate memory when no more is available	

3.3.7.1.9.1.9 LLCSC DESIGN

None.

3.3.7.1.9.1.10 UNIT DESIGN

None.

3.3.7.1.9.2 AVAILABLE SPACE LIST OPERATIONS PACKAGE DESIGN

This package contains a set of routines used to manipulate an available space list which is maintained local to the part instantiating this package.

The first routine, New Node, will return a node to the calling routine. If a node is available in the available space list, the node will be retrieved from there. If not, a new node will be dynamically allocated. If no memory is available for the allocation, a STORAGE ERROR exception is raised.

The second routine, Save Node, places a node in the available space list.

The third routine, Save_List, places a list of nodes in the available space list.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.9.2.1 REQUIREMENTS ALLOCATION

This part helps meet CAMP requirements R164, R165, R167.

3.3.7.1.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this was specified in the package body of Abstract_Data_Structures.

Page 1736

Data types:

The following table summarizes the generic formal types required by this part:

Name | Type | Description |

	Name	туре		ļ
	Nodes Pointers 	limited private access Nodes	A single element in the available space list A pointer to an element in the available space list	

Data objects:

The following table summarizes the generic formal objects required by this part. All of these objects are in/out parameters and are changed by calls to the enclosed routines.

	Name	Туре	Value	Description	
	Available_ Length	INTEGER	N/A 	Length of the available space list	
	Available_Head	Pointers	N/A	Points to the first element in the available space list	
	Available_Tail	Pointers	N/A	Points to the last element in the available space list	

Subprograms:

The following table describes the generic formal subprograms required by this part:

	Name		Туре		Description	
	Dot_Next		function		Given a pointer to a node, this function returns a pointer to the next node in the list	.
İ	Set_Next	ĺ	procedure	Ì	Given two points, A and B, sets A.Next equal to B	j

3.3.7.1.9.2.4 LOCAL DATA

None.

3.3.7.1.9.2.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.2.6 PROCESSING

The following describes the processing performed by this part:

separate (Abstract_Data_Structures)
package body Available_Space_List_Operations is

end Available_Space_List_Operations;

3.3.7.1.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.7.1.9.2.8 LIMITATIONS

None.

3.3.7.1.9.2.9 LLCSC DESIGN

None.

٢

3.3.7.1.9.2.10 UNIT DESIGN

3.3.7.1.9.2.10.1 NEW NODE UNIT DESIGN

This function returns a node to the calling routine. If nodes are available in the space list, the node returned will be from there. If the available space list is empty, this routine will attempt to dynamically allocate memory. If no more memory is available on the system, a STORAGE_ERROR exception will be raised.

3.3.7.1.9.2.10.1.1 REQUIREMENTS ALLOCATION

This part helps meets CAMP requirements R164, R164, R176.

3.3.7.1.9.2.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.2.10.1.3 INPUT/OUTPUT

None.

3.3.7.1.9.2.10.1.4 LOCAL DATA

Data objects:



The following table describes the data objects maintained by this part:

Name		Туре	Value	Description	
Ptr New_Ava Head	ilable_	Pointers Pointers	N/A N/A 	Points to the node being returned Temporary variable used to mark where Available Head will point when this routine is exited	

3.3.7.1.9.2.10.1.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.2.10.1.6 PROCESSING

The following describes the processing performed by this part:

function New Node return Pointers is

> Ptr : Pointers; New_Available_Head : Pointers;

begin

if Available Length > 0 then

-- -- initialize node being returned Set_Next (Ptr => Ptr, Ptr_dot_Next => NULL);

else

 --allocate space to get the node Ptr := NEW Nodes;

end if;

Ř

8

Page 1739

```
return Ptr;
```

end New_Node;

3.3.7.1.9.2.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table describes the subroutines required by this part and defined as generic formal subprograms to the Available Space List Operations package:

	Name		Туре		Description	
	Dot_Next		function		Given a pointer to a node, this function returns a pointer to the next node in the list	
İ	Set_Next	İ	procedure	'I	Given two points, A and B, sets A.Next equal to B	1

Data types:

6

The following table summarizes the types required by this part and defined as generic formal parameters to the Abstract_Data_Structures. Available_Space_-List_Operations package.

	Name		Туре		Description	Ī
	Nodes Pointers		limited private access Nodes		A single element in the available space list A pointer to an element in the available space list	

Data objects:

The following table summarizes the objects required by this part and defined as generic formal parameters to the Abstract_Data_Structures. Available_Space_-List Operations package.

Name	Туре	Value	Description	
Available_ Length	INTEGER	N/A	Length of the available space list	
Available_Head	Pointers	N/A	Points to the first element in the available space list	

Page 1740

3.3.7.1.9.2.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

 Name
 | When/Why Raised
 |

 | STANDARD.STORAGE_ERROR
 | Raised if an attempt is made to allocate memory
 |

 | When no more is available
 |

3.3.7.1.9.2.10.2 SAVE NODE UNIT DESIGN

This procedure returns a node to the available space list. The node returned to the list is the one pointed to by Saved_Node.

3.3.7.1.9.2.10.2.1 REQUIREMENTS ALLOCATION

This part helps meets CAMP requirements R164, R164, R176.

3.3.7.1.9.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.2.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	
	Saved_Node		Pointers		in		Pointer to the node which is to be placed in the available space list	

3.3.7.1.9.2.10.2.4 LOCAL DATA

None.

3.3.7.1.9.2.10.2.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.2.10.2.6 PROCESSING

The following describes the processing performed by this part:

procedure Save Node(Saved Node : in Pointers) is

```
begin
```

end Save_Node;

3.3.7.1.9.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table describes the subroutines required by this part and defined as generic formal subprograms to the Available_Space_List_Operations package:

```
10
```

	Name	I	Туре		Desci	ipt	ion									Ī
	Set_Next	1	procedure		Given	two	points,	A	and	В,	sets	A.Next	equal	to	B	

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Abstract_Data_Structures. Available_Space_List_Operations package.

	Name		Туре		Description	Ī
	Nodes Pointers		limited private access Nodes		A single element in the available space list A pointer to an element in the available space list	

Data objects:

The following table summarizes the objects required by this part and defined as generic formal parameters to the Abstract_Data_Structures. Available_Space_-List Operations package.

	Name	Туре	Value	Description	
	Available_ Length	INTEGER	N/A N/A	Length of the available space list	
				available space list	

3.3.7.1.9.2.10.2.8 LIMITATIONS

None.

3.3.7.1.9.2.10.3 SAVE LIST UNIT DESIGN

This procedures places a linked list of nodes in the available space list.

3.3.7.1.9.2.10.3.1 REQUIREMENTS ALLOCATION

None.

3.3.7.1.9.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.2.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Mode		Description	
	Saved_Head	Pointers	in		Pointer to the first node to be placed in the available space list	
Ì	Saved_Tail	Pointers	in		Pointer to the last node to be placed in the available space list	ĺ
ļ	Node_Count	POSITIVE	in		Number of nodes to be placed in the available space list	

3.3.7.1.9.2.10.3.4 LOCAL DATA

None.

Page 1742

3.3.7.1.9.2.10.3.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.2.10.3.6 PROCESSING

The following describes the processing performed by this part:

begin

end Save List;

3.3.7.1.9.2.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table describes the subroutines required by this part and defined as generic formal subprograms to the Available Space List Operations package:

1	Name		Туре		Desci	ipt	ion									
	Set_Next	Ι	procedure		Given	two	points,	A	and	в,	sets	A.Next	equal	to	В	Ī

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Available_Space_List_Operations LLCSC:

	Name	Туре	Description	
	Pointers	access	Nodes A pointer to an element in the available space list	



S.

Data objects:

The following table summarizes the objects required by this part and defined as generic formal parameters to the Available Space List Operations LLCSC:

Name	Туре	Value	Description	
Available_ Length	INTEGER	N/A	Length of the available space list	
Available_Tail	Pointers	N/A	Points to the last element in the available space list	

3.3.7.1.9.2.10.3.8 LIMITATIONS

None.

3.3.7.1.9.3 BOUNDED FIFO BUFFER PACKAGE DESIGN (CATALOG #P331-0)

This generic package defines the data type and contains the operations required to perform first-in-first-out buffering operations on incoming data. The head always points to a dummy node. The first node following the dummy node contains the next piece of data to be retrieved. The tail always points to where the next element should be added. If the tail points to the element immediately in front of the head, the buffer is empty. If the tail points to the same element as the head, the buffer is full. Since the buffer is implemented as an array, the head and tail will advance through the array in a circular fashion, but no overwriting of data currently in the buffer will be permitted.

Empty FIFO buffer: +-+ <-----Head +-+ +-+ <----Tail +-+ +-+ +-+

Full FIFO buffer: Tail---->+-+ <-----Head +-+ +-+ +-+ +-+ +-+ +-+

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP required R125.

3.3.7.1.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of the Abstract Data_Structures package:

Data types:

The following table summarizes the generic formal types required by this part:

	Name		Туре		Desc	cription							*****	
	Elements		private		User	defined	type	of	data	contained	in	the	buffer	

Data objects:

The following table summarizes the generic formal objects required by this part:

	Name		Туре		Value		Description	
	Initial_ Buffer_Size		POSITIVE		N/A	1	Maximum number of elements which can be in the buffer at any given time	

3.3.7.1.9.3.4 LOCAL DATA

None.

3.3.7.1.9.3.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.3.6 PROCESSING

The following describes the processing performed by this part:

separate (Abstract_Data_Structures)
package body Bounded FIFO Buffer is

end Bounded FIFO Buffer;

3.3.7.1.9.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table describes the data types which were previously defined in this part's specification:

	Name	Туре	Range	Description	1
	Buffer_Range Buffer_ Statuses	NATURAL subtype discrete type	0 Buffer_Size Empty, Available, Full	Used to dimension the list of elements Used to indicate the status of the buffer	

The following table describes the data types defined in the private part of this part's specification:

1	Name		Туре		Range		Description	
1	Buffers		record		N/A		List of data along with relevant information	
İ	Lists	İ	array	İ	N/A	i	Array of elements	İ

Data objects:

The following table describes the data objects which were previously defined in this part's specification:

	Name Type		Value	Description	
	Buffer_Size	POSITIVE	Initial_ Buffer_Size	Number of usable elements in a buffer	

Exceptions:

The following table describes the exceptions which were previously defined in this part's specification:

1	Name		Description	Ī
	Buffer_Empty Buffer_Full	1	Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer Error condition raised if an attempt is made to add elements to a full buffer	

3.3.7.1.9.3.8 LIMITATIONS

None.

Page 1746

R.

```
3.3.7.1.9.3.9 LLCSC DESIGN
```

None.

3.3.7.1.9.3.10 UNIT DESIGN

3.3.7.1.9.3.10.1 CLEAR BUFFER UNIT DESIGN

This procedure clears an input buffer by setting its length to 0 and resetting its head and tail to 0 and 1, respectively.

3.3.7.1.9.3.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R125.

3.3.7.1.9.3.10.1.2 LOCAL ENTITIES DESIGN

None.

(

3.3.7.1.9.3.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре	Mode	Description	
Buffer	Buffers	out	FIFO buffer being accessed	

3.3.7.1.9.3.10.1.4 LOCAL'DATA

None.

3.3.7.1.9.3.10.1.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.3.10.1.6 PROCESSING

The following describes the processing performed by this part:

procedure Clear Buffer (Buffer : out Buffers) is

-- -- declaration section

Buffer_Length : Buffer_Range renames Buffer.Buffer_Length;

Page 1748

.

```
Head
                   : Buffer Range renames Buffer.Head;
      Tail
                   : Buffer Range renames Buffer.Tail;
__ _______
-- -- begin procedure Clear Buffer
begin
      Buffer Length := 0;
      Head
                   := 0;
      Tail
                   := 1;
   end Clear Buffer ;
3.3.7.1.9.3.10.1.7 UTILIZATION OF OTHER ELEMENTS
None.
3.3.7.1.9.3.10.1.8 LIMITATIONS
None.
3.3.7.1.9.3.10.2 ADD ELEMENT UNIT DESIGN
This procedure adds an element to an input buffer if the buffer is not already
full. After the element is added, the tail is advanced one place in the buffer
and the length counter is incremented by 1.
The exception Buffer_Full is raised if an attempt is made to add an element to
an already full buffer.
3.3.7.1.9.3.10.2.1 REQUIREMENTS ALLOCATION
This part meets CAMP requirement R125.
3.3.7.1.9.3.10.2.2 LOCAL ENTITIES DESIGN
None.
3.3.7.1.9.3.10.2.3 INPUT/OUTPUT
FORMAL PARAMETERS:
The following table describes this part's formal parameters:
```

8

1	Name		Туре	1	Mode		Description	
	Buffer New_Element		Buffers Elements		in out in		FIFO buffer being accessed Element to be added to the buffer	

3.3.7.1.9.3.10.2.4 LOCAL DATA

None.

_ _

- -

3.3.7.1.9.3.10.2.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.3.10.2.6 PROCESSING

The following describes the processing performed by this part:

procedure Add Element (New Element : in Elements; Buffer : in out Buffers) is

--declaration section --_____

> renames Buffer.List; List : Lists Buffer Length : Buffer Range renames Buffer.Buffer_Length; : Buffer Range renames Buffer.Head; Head : Buffer Range renames Buffer.Tail; Tail

_____ -- --begin procedure Add Element _____

begin

```
--make sure buffer isn't full
   if Head = Tail then
      raise Buffer_Full;
   end if;
   List(Tail) := New Element;
   Buffer_Length := Buffer_Length + 1;
   if Tail = Buffer Size then
      Tail := 0;
   else
      Tail := Tail + .1;
   end if;
end Add Element ;
```

Page 1750

83

3.3.7.1.9.3.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal types to the Abstract_Data_Structures. Bounded_FIFO_Buffer package:

	Name		Туре		Description								
1	Elements	1	private		User defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the package specification of Abstract Data Structures. Bounded FIFO Buffer.

Ī	Name		Туре	1	Range	·	Description	-
	Buffer_Range		NATURAL subtype		0 Buffer_Size		Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Bounded_FIFO_Buffer package:

	Name		Туре		Range		Description	
	Buffers		record		N/A		List of data along with relevant information	
İ	Lists	İ	array	Ì	N/A	İ	Array of elements	Ì

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_FIF0_Buffer:

Name			Туре		Value		Description	-
	Buffer_Size		POSITIVE		Initial Buffer_Size		Number of usable elements in a buffer	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract Data Structures. Bounded FIFO Buffer: Ć

6

Name	Description	
Buffer_Ful:	Error condition raised if an attempt is made to add elements to a full buffer	

3.3.7.1.9.3.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	Description	ī
	Buffer_Full	Error condition raised if an attempt is made to add elements to a full buffer	

3.3.7.1.9.3.10.3 RETRIEVE ELEMENT UNIT DESIGN

This procedure retrieves the top element in the buffer if the buffer is not empty. The head is advanced through the buffer by 1 before the element is retrieved and the size of the buffer is decremented by 1 after the element is retrieved.

If the buffer is empty before calling this routine, the exception Buffer_Empty is raised.

3.3.7.1.9.3.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R125.

3.3.7.1.9.3.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.3.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Mode	Ī	Description	1
	Buffer Old_Element	Buffers Elements	in out out		FIFO buffer being accessed · Element retrieved from the buffer	

3.3.7.1.9.3.10.3.4 LOCAL DATA

None.

3.3.7.1.9.3.10.3.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.3.10.3.6 PROCESSING

The following describes the processing performed by this part:

procedure Retrieve Element (Buffer : in out Buffers; Old Element : out Elements) is _____ --declaration section --_____ ___ Buffer Length : Buffer Range renames Buffer.Buffer Length; Buffer_Range renames Buffer.Head; Head List : Lists renames Buffer.List; : Buffer_Range_renames Buffer.Tail; Tail -- --begin procedure Retrieve Element begin

-- --make sure don't have an empty buffer
if Head = (Tail-1) or else (Tail = 0 and Head = Buffer_Size) then
raise Buffer_Empty;
end if;
if Head = Buffer_Size then
Head := 0;
else
Head := Head + 1;
end if;
Old Element := List(Head);
Buffer_Length := Buffer_Length - 1;
end Retrieve Element ;

3.3.7.1.9.3.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Page 1752

8

Data types:

18

5

The following table summarizes the types required by this part and defined as generic formal types to the Abstract_Data_Structures. Bounded_FIFO_Buffer package:

	Name		Туре		Description								
	Elements	I	private		User defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the package specification of Abstract Data Structures. Bounded FIFO Buffer.

	Name	Type	Range	Description	
	Buffer_Range	NATURAL subtype	0 Buffer_Size	Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Bounded_FIFO_Buffer package:

Ī	Name	Туре	j Range		Description	
-	Buffers	record	N/A		List of data along with relevant	
1	Lists	array	N/A	İ	Array of elements	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract Data Structures. Bounded FIFO Buffer:

	Name	Description	
	Buffer_Empty	Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer	

3.3.7.1.9.3.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	Description	
Buffer_Empty	Error condition raised if an attempt is made to look at retrieve elements from an empty buffer	or

3.3.7.1.9.3.10.4 PEEK UNIT DESIGN

This function returns the first element of the buffer if the buffer is not empty. The status of the buffer is not changed, however, and the element itself remains in the buffer.

The Buffer_Empty exception is raised if an attempt is made to look at an empty buffer.

3.3.7.1.9.3.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R125.

3.3.7.1.9.3.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.3.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	1	Туре	1	Mode		Descr	iption			
1	Buffer	1	Buffers	1	in out	I	FIFO b	ouffer	being	accessed	l

3.3.7.1.9.3.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained local to this part:

1	Name		Туре	Ι	Description	
	Spot	1	Buffer_Range	I	Marks location of element to be looked at	

3.3.7.1.9.3.10.4.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.3.10.4.6 PROCESSING

The following describes the processing performed by this part:

function Peek (Buffer : in Buffers) return Elements is

Page 1754

```
_____
    --declaration section
--
    _____
    Buffer Length : Buffer Range renames Buffer.Buffer Length;
    Head : Buffer_Range renames Buffer.Head;
              : Buffer Range renames Buffer.Tail;
     Tail
    List
              : Lists renames Buffer.List;
    Spot
               : Buffer Range;
_ ____
-- -- begin function Peek
__ ______
  begin
     --make sure don't have an empty buffer
--
    if Head = (Tail-1) or else (Tail = 0 and Head = Buffer Size) then
       raise Buffer Empty;
    end if;
    if Head = Buffer Size then
       Spot := 0;
    else
       Spot := Head + 1;
    end if;
    return List(Spot);
  end Peek ;
3.3.7.1.9.3.10.4.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:
The following tables describe the elements used by this part but defined
elsewhere in the parent top level component:
Data types:
The following table summarizes the types required by this part and defined as
generic formal types to the Abstract Data Structures. Bounded FIFO Buffer
package:
                  Name | Type | Description
.
| Elements | private | User defined type of data contained in the buffer |
____________
```

The following table summarizes the types required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_FIFO_Buffer.

Page 1755

1	Name		Туре		Range		Description	1
 	Buffer_Range		NATURAL subtype	0) Buffer_Size	[Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Bounded_FIF0_Buffer package:

1	Name		Туре		Range		Description	1
	Buffers		record		N/A		List of data along with relevant	
	Lists	i	array	i i	N/A	ľ	Array of elements	İ

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_FIFO_Buffer:

	Name	Description	Ī
	Buffer_Empty	Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer	

3.3.7.1.9.3.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	Description	-
	Buffer_Empty	Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer	

3.3.7.1.9.3.10.5 BUFFER STATUS UNIT DESIGN

This function returns the status of the buffer. If there are no elements in the buffer, the status is empty; if there is no room for additional elements, the status is full; otherwise, the status is available.

3.3.7.1.9.3.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R125.

Page 1756

D

Q

3.3.7.1.9.3.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.3.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	1	Туре		Mode		Des	cription	 1		
	Buffer		Buffers		in out		FIFO	buffer	being	accessed	Ī

3.3.7.1.9.3.10.5.4 LOCAL DATA

Data objects:

Ć

•

The following objects are maintained local to this part:

Status Buffer_Statuses Status of the buffer

3.3.7.1.9.3.10.5.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.3.10.5.6 PROCESSING

The following describes the processing performed by this part:

function Buffer Status (Buffer : in Buffers) return Buffer Statuses is

-- --declaration section -- --declaration section -- Head : Buffer_Range renames Buffer.Head; Tail : Buffer_Range renames Buffer.Tail; Status : Buffer_Statuses; -- --begin function Buffer_Status

begin

```
if Head = (Tail-1) or else (Tail = 0 and Head = Buffer_Size) then
   Status := Empty;
elsif Head = Tail then
   Status := Full;
else
   Status := Available;
end if;
return Status;
end Buffer_Status ;
```

3.3.7.1.9.3.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_FIFO_Buffer.

1	Name	Туре	Range	Description	
	Buffer_Range Buffer_ Statuses	NATURAL subtype discrete type	0 Buffer_Size Empty, Available, Full	Used to dimension the list of elements Used to indicate the status of the buffer	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Bounded_FIF0_Buffer package:

1	Name		Туре		Range		Description	Ī
	Buffers		record		N/A		List of data along with relevant information	

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_FIF0_Buffer:

	Name	Ι	Туре		Value		Description	
	Buffer_Size		POSITIVE		Initial_ Buffer_Size		Number of usable elements in a buffer	

Page 1758

```
3.3.7.1.9.3.10.5.8 LIMITATIONS None.
```

3.3.7.1.9.3.10.6 BUFFER_LENGTH UNIT DESIGN

This function returns the length of the current buffer.

3.3.7.1.9.3.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R125.

3.3.7.1.9.3.10.6.2 LOCAL ENTITIES DESIGN

3.3.7.1 2 3.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Náme	Type	Mode Description	
Buffer	Buffers	in out FIFO buffer being accessed	

```
3.3.7.1.9.3.10.6.4 LOCAL DATA
```

None.

3.3.7.1.9.3.10.6.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.3.10.6.6 PROCESSING

The following describes the processing performed by this part:

function Buffer_Length (Buffer : in Buffers) return Buffer_Range is

begin

return Buffer.Buffer_Length;

end Buffer_Length ;

3.3.7.1.9.3.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of Abstract Data Structures. Bounded FIFO Buffer.

1	Name	Ι	Туре		Range	I	Description	1
	Buffer_Range		NATURAL subtype		0 Buffer_Size		Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Bounded_FIF0_Buffer package:

1	Name		Туре		Range		Description	
	Buffers		record		N/A		List of data along with relevant information	

3.3.7.1.9.3.10.6.8 LIMITATIONS

None.

3.3.7.1.9.4 UNBOUNDED FIFO BUFFER PACKAGE DESIGN (CATALOG #P332-0)

This generic package defines the data type and contains the operations required to perform first-in-first-out buffering operations on incoming data. The head of the buffer always points to a dummy node. The first node following the dummy node contains the next piece of data to be retrieved. The tail always points to the node containing the last element added to the buffer. If the tail points to the same node as the head, the buffer is empty.

A buffer must be initialized before it is used. If an attempt is made to use an uninitialized buffer, the exception Buffer Not Initialized will be raised. The Initialized Buffer procedure returns an initialized buffer. The Clear -Buffer procedure returns the nodes of a buffer to the available space list and then returns an initialized buffer.

An available space list is maintained local to this part. When this part is elaborated the available space list will have a dummy node plus Initial -Available_Space_Size nodes. When nodes are added to the buffer, the Add -Element routine will try to get a node from the available space list before attempting to allocate more memory. When the Retrieve Element routine is called, the unused node will be returned to the available space list for later use. The memory committed to the available space may be deallocated by calling
88) 8

Ö

the Free Memory procedure.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.9.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.3.7.1.9.4.2 LOCAL ENTITIES DESIGN

Data structures:

An available space list is maintained local to this part's package body.

Subprograms:

The following subprograms are contained local to this body:

Name	Туре	Description	1
Free Node Dot_Next	procedure function	Instantiation of UNCHECKED DEALLOCATION Given a pointer P, this function returns the value of P.Next	
Set_Next	procedure	Given two points P & Q, this procedure sets P.Next = Q	Ì

The following subprograms are contained in this part as a result of renaming operations on identically named routines contained in the locally instantiated Available_Space_Operations package.

-	Name	Туре	Description	
	New_Node Save_Node Save_List	function procedure procedure	Returns a node to the calling routine; will get a node from the available space list if possible, otherwise will allocate a new node Handles placing a node in the available space list Handles placing a list of nodes in the available space list	t

This package body contains code to initialize the Available Space List. This code is executed when the package is elaborated. At a minimum, this code calls the Initialize Buffer procedure to initialize the Available Space List so it contains a dummy node. If the generic formal object Initial Available Space – Size is greater than or equal to 1, this routine then places the requested number of nodes (in addition to the dummy node) in the available space list.

Page 1762

3.3.7.1.9.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of the Abstract_Data_Structures package:

Data types:

The following table summarizes the generic formal types required by this part:

1	Name		Туре		Desc	ription								
I	Elements		private		User	defined	type	of	data	contained	in	the	buffer	

Data objects:

The following table summarizes the generic formal objects required by this part:

1	Name	Туре		Description	
	Initial_Available_ Space_Size	NATURAL		Number of nodes to be initially placed in the available space list	

3.3.7.1.9.4.4 LOCAL DATA

None.

3.3.7.1.9.4.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.4.6 PROCESSING

The following describes the processing performed by this part:

with UNCHECKED_DEALLOCATION; separate (Abstract_Data Structures) package body Unbounded_FIFO_Buffer is

-- --declaration section

-- -- this variable is accessed ONLY when setting up the available space list Initial Head : Pointers := new Nodes;

Available_Space : Buffers := (Current_Length => 0, Head => Initial Head,

Tail => Initial Head); Available Length : INTEGER renames Available Space.Current Length; Available Head : Pointers renames Available Space.Head; Available Tail : Pointers renames Available Space.Tail; procedure Free is new UNCHECKED DEALLOCATION (Object => Nodes, Name => Pointers); procedure Free Node (Which Node : in out Pointers) renames Free; function Dot Next (Ptr : in Pointers) return Pointers; procedure Set Next (Ptr : in Pointers; Ptr dot Next : in Pointers); package Available Space Operations is new Available Space List Operations (Nodes => Nodes, Pointers => Pointers, Available Length => Available Length, Available Head => Available Head, Available Tail => Available Tail); function New Node return Pointers renames Available Space Operations.New Node; procedure Save Node (Saved Node : in Pointers) renames Available Space Operations. Save Node; procedure Save List (Saved Head : in Pointers; Saved Tail : in Pointers; Node Count : in POSITIVE) renames Available Space Operations.Save List; --begin package Unbounded FIFO Buffer -- (see header for package body for details) begin -- -- set up available space list if one is desired if Initial Available Space Size > 0 then Add_Nodes_To_Available_Space_List: for I in I..Initial Available Space Size loop Available Tail.Next := NEW Nodes; Available Tail := Available Tail.Next; end loop Add Nodes to Available Space List; Available Length := Initial Available Space Size;

end if;

50

Page 1763

end Unbounded FIFO Buffer;

3.3.7.1.9.4.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part: 1. Unchecked Deallocation

Subprograms and task entries:

The following table describes the subroutines required by this part:

1	Name		Туре	1	Source		Description	
	UNCHECKED DEALLOCATION		generic function		N/A		Used to deallocate memory	

Exceptions:

The following table describes the exceptions required by this part and defined in the Ada predefined package STANDARD:

	Name	Description	
	STORAGE_ERROR	Raised when an attempt is made to dynamically allocate more memory than is available	

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table describes the packages required by this part and specified in the package body of the Abstract_Data_Structures package:

	Name		Туре		Description	1
 	Available_Space_ List_Operations		generic package		Contains the routines required to retrieve a node from and place a node in the available space list	

Data types:

The following data types were previously defined in this part's package specification:

Page 1764





	Name		Туре		Ran 'e		Description
	Buffer_ Statuses		discrete type		Empty, Available, Uninitialized		Used to indicate the status of the buffer

The following data types were previously defined in the private portion of this part's package specification:

1	Name	Туре	Range		Description	
	Nodes	record	N/A		A single entity in the buffer; contains data and a pointer to the next node	
İ	Pointers	access	N/A	İ	Points to a node in the buffer	i
	Buffers	record	N/A		Record containing the value of the current length, head, and tail of the buffer	

Exceptions:

The following exceptions were previously defined in this part's package specification:

```
50
```

 Name
 Description

 Buffer_Empty
 Error condition raised if an attempt is made to look at or

 retrieve elements from an empty buffer

 Buffer_Not
 Raised if an attempt is made to use an uninitialized buffer

 Initialized

3.3.7.1.9.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why Raised	Ī
	Storage_Error	Raised during elaboration of this package if an attempt is made to allocate memory when no more is available	

3.3.7.1.9.4.9 LLCSC DESIGN

None.

3.3.7.1.9.4.10 UNIT DESIGN

3.3.7.1.9.4.10.1 INITIALIZE BUFFER UNIT DESIGN

This procedure initializes a buffer. It does this in the following manner:

1) If the buffer has never been initialized then: o places a dummy node in the buffer and o initializes the length to 0

2) else if the buffer has elements in it then: o calls the Clear_Buffer procedure

3) else if the buffer has a length of 0 then o does nothing

3.3.7.1.9.4.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.3.7.1.9.4.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.4.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	1	Туре		Mode	I	Desc	ription	1		1
	Buffer	1	Buffers		in out		FIFO	buffer	being	initialized	1

3.3.7.1.9.4.10.1.4 LOCAL DATA

None.

3.3.7.1.9.4.10.1.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.4.10.1.6 PROCESSING

The following describes the processing performed by this part:

procedure Initialize Buffer (Buffer : in out Buffers) is

-- -- declaration section

Page 1766

```
Current Length : INTEGER renames Buffer.Current Length;
     Head : Pointers renames Buffer.Head;
     Tail
                   : Pointers renames Buffer.Tail:
     _____
-- --begin procedure Initialize Buffer
begin
     if Current Length = -1 then
        --handle an uninitialized buffer
_ _
        Head := New Node;
        Tail := Head;
        Current Length := 0;
     elsif Current Length > 0 then
        --handle a buffer that has something in it
        Clear Buffer(Buffer => Buffer);
     else
        --current length = 0 so it is already initialized
        NULL:
     end if;
  end Initialize Buffer ;
3.3.7.1.9.4.10.1.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:
The following tables describe the elements used by this part but defined
elsewhere in the parent top level component:
Subprograms and task entries:
The following table summarizes the subroutines and task entries required by
this part and defined in the package specification of Unbounded FIFO Buffer:
```

1	Name	Туре	Description				
	Clear_ Buffer	procedure 	Returns all space list	the nodes	in a buffer	to the availab	le

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

+1

1	Name	Туре	Range	Description	
	Nodes	record	N/A	A single entity in the buffer; contains data and a pointer to the next node	
İ	Pointers	access	N/A	Points to a node in the buffer	İ
	Buffers	record	N/A	Record containing the value of the current length, head, and tail of the buffer	

3.3.7.1.9.4.10.1.8 LIMITATIONS

None.

3.3.7.1.9.4.10.2 CLEAR BUFFER UNIT DESIGN

This procedure returns all the elements in a buffer, except for the dummy node, to the available space list. If this routine is sent an uninitialized buffer, a Buffer_Not_Initialized exception is raised.

3.3.7.1.9.4.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.3.7.1.9.4.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.4.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Type	Mode		Description
Ī	Buffer	Buffers	in out	1	FIFO buffer being cleared

3.3.7.1.9.4.10.2.4 LOCAL DATA

Data objects:

The following table describes the objects maintained local to this part:

	Name		Туре		Description	
1	This_Node		Pointers		Node to be placed in the available space list	

Page 1768

4

```
3.3.7.1.9.4.10.2.5 PROCESS CONTROL
Not applicable.
```

3.3.7.1.9.4.10.2.6 PROCESSING

The following describes the processing performed by this part:

procedure Clear Buffer (Buffer : in out Buffers) is

--_______ --declaration section ___ **** --Current_Length : INTEGER renames Buffer.Current_Length; : Pointers renames Buffer.Head; : Pointers renames Buffer.Tail; Head Tail This Node : Pointers; -- -- begin procedure Clear Buffer begin --make sure this is an initialized buffer ---if Current Length = -1 then raise Buffer Not Initialized; end if; --placed nodes in the available space list --Save List (Saved Head => Head.Next, Saved Tail => Tail, Node Count => Current Length); --reinitialize buffer variables ---Current Length := 0; Head.Next := NULL; Tail := Head:

end Clear_Buffer ;

3.3.7.1.9.4.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded_FIFO_Buffer:

	Name		Туре		Description	Ī
	Save_List		procedure	:	Handles placing a list of nodes in the available space list	

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

	Name	Type	Range	Description	
	Nodes	record	N/A	A single entity in the buffer; contains data and a pointer to the next node	
I.	Pointers	access	N/A	Points to a node in the buffer	İ
	Buffers	record	N/A	Record containing the value of the current	
		l 		length, head, and tail of the buffer	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Unbounded_FIFO_-Buffer:

	Name		Descri	pt	ion	********						****		
	Buffer_Not_ Initialized		Raised	if	an	attempt	is	made	to	use	an	uninitialized	buffer	

3.3.7.1.9.4.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

1	Name	When/Why Raised	
	Buffer_Not_Initialized	Raised if an attempt is made to use an uninitialized buffer	

3.3.7.1.9.4.10.3 FREE MEMORY UNIT DESIGN

This procedure deallocates the memory occupied by the available space list.

Page 1770

Ŋ

3.3.7.1.9.4.10.3.1 REQUIREMENTS ALLOCATION This part meets CAMP requirement R164.

3.3.7.1.9.4.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.4.10.3.3 INPUT/CUTPUT None.

3.3.7.1.9.4.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name		Туре		Value		Description	
	Node_to_be_Freed		Pointers		N/A	 	Pointer to the node to be deallocated	

3.3.7.1.9.4.10.3.5 PROCESS CONTROL

Not applicable.

6

3.3.7.1.9.4.10.3.6 PROCESSING

The following describes the processing performed by this part:

procedure Free Memory is

-- -- declaration section

Node_to_be_Freed : Pointers;

-- --begin procedure Free_Memory

begin

Clear Out Available Space List: while Available Head /= Available Tail loop

> Node To Be Freed := Available Head; Available Head := Available Head.Next;

Page 1772

666

Free_Node (Which_Node => Node_to_be_Freed);

end loop Clear Out Available Space List;

Available Length := 0;

end Free_Memory ;

3.3.7.1.9.4.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded FIFO Buffer:

<u> </u>	Name		Туре	1	Description	
	Free_Node		procedure		Instantiation of UNCHECKED_DEALLOCATION	1

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Abstract_Data_Structures. Unbounded_FIFO_Buffer package:

1	Name		Туре		Dese	cription								
	Elements		private	1	User	defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

	Name	Туре	Range	Description	
	Nodes	record	N/A	A single entity in the buffer; contains data and a pointer to the next node	
	Pointers Buffers	access record	N/A N/A	Points to a node in the buffer Record containing the value of the current length, head, and tail of the buffer	

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Unbounded FIFO Buffer:

1	Name		Туре		Description	
	Available_ Space		Buffers		List of available nodes; nodes will be added to list when Retrieve_Element is called and retrieved from the list when Add_Element is called; the nodes in the list are deallocated when Clear_Memory is called	

The following table summarizes the data objects required by this part and defined in the package body of Unbounded_FIFO_Buffer:

	Name		Туре		Value		Description	
	Available_ Length Available_ Head Available_ Tail		INTEGER Pointers Pointers		Available Space. Current_Length Available_Space. Head Available_Space. Tail		Indicates the current length of the available space list Points to the head node in the available space list Points to the tail node in the available space list	

3.3.7.1.9.4.10.3.8 LIMITATIONS

None.

(

121

3.3.7.1.9.4.10.4 ADD_ELEMENT UNIT DESIGN

This procedure adds an element to the end of the FIFO buffer.

If the buffer has not been initialized, the exception Buffer_Not_Initialized is raised.

The Storage Error exception is raised if a call to this routine requires memory to be dynamically allocated when no more memory is available.

3.3.7.1.9.4.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.3.7.1.9.4.10.4.2 LOCAL ENTITIES DESIGN

None.

ф)

3.3.7.1.9.4.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

 Name
 Type
 Mode
 Description
 |

 Buffer
 Buffers
 in out
 FIFO buffer being accessed
 |

 New_Element
 Elements
 in
 Element to be added to the buffer
 |

3.3.7.1.9.4.10.4.4 LOCAL DATA

None.

3.3.7.1.9.4.10.4.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.4.10.4.6 PROCESSING

The following describes the processing performed by this part:

procedure Add_Element (New_Element : in Elements; Buffer : in out Buffers) is

-- -- declaration section

> Current_Length : INTEGER renames Buffer.Current_Length; Tail : Pointers renames Buffer.Tail;

New Tail : Pointers;

-- --begin procedure Add_Element

begin

- -- --make sure buffer has been initialized
 if Current_Length = -1 then
 raise Buffer_Not_Initialized;
 end if;
- -- --now get a node
 New_Tail := New_Node;
- -- -- now adjust the buffer Tail.Next := New_Tail; Tail := New_Tail;

Page 1774

S.

Tail.Data := New_Element; Current Length := Current Length + 1;

end Add_Element ;

3.3.7.1.9.4.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded_FIFO_Buffer:

	Name		Туре		Description	
	New_Node		function		Returns a node to the calling routine; will get a node from the available space list if possible, otherwise will allocate a new node	

Data types:

So

The following table summarizes the types required by this part and defined as generic parameters to the Abstract_Data_Structures. Unbounded_FIFO_Buffer package:

	Name		Туре		Des	cription								
	Elements		private	{	User	defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

1	Name		Туре		Range		Description	Ī
 	Nodes Pointers		record access		N/A N/A		A single entity in the buffer; contains data and a pointer to the next node Points to a node in the buffer	
ļ	Buffers	ļ	record	İ	N/A	İ	Record containing the value of the current length, head, and tail of the buffer	İ

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Unbounded_FIFO_-Buffer:

1	Name		Descri	pti	on								******	1
	Buffer_Not_ Initialized		Raised	if a	an	attempt	is	made	to	use	an	uninitialized	buffer	

The following table describes the exceptions required by this part and defined in the Ada predefined package STANDARD:

	Name	Description	
	STORAGE_ERROR	Raised when an attempt is made to dynamically allocate more memory than is available	

3.3.7.1.9.4.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why Raised	
	Storage_Error Buffer_Not_Initialized	Raised if an attempt is made to allocate memory when no more is available Raised if an attempt is made to use an uninitialized buffer	

3.3.7.1.9.4.10.5 RETRIEVE ELEMENT UNIT DESIGN

This procedure retrieves the oldest element from the FIFO buffer, places the spare node on the available space list, and updates the status of the FIFO buffer.

If the buffer has not been initialized, a Buffer_Not_Initialized exception is raised.

If the buffer is empty, a Buffer Empty exception is raised.

3.3.7.1.9.4.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.3.7.1.9.4.10.5.2 LOCAL ENTITIES DESIGN

None.

ŝ

3.3.7.1.9.4.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	
	Buffer Old_Element		Buffers Elements		in out out	F E	IFO buffer being accessed lement retrieved from the buffer	

3.3.7.1.9.4.10.5.4 LOCAL DATA

Data objects:

The following table describes the objects maintained local to this part:

1	Name		Туре		Description	
	This_Node	1	Pointers		Node to be placed in the available space list	

3.3.7.1.9.4.10.5.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.4.10.5.6 PROCESSING

The following describes the processing performed by this part:

procedure Retrieve_Element (Buffer : in out Buffers; Old Element : out Elements) is

-- -- declaration section

Current_Length : INTEGER renames Buffer.Current_Length; Head : Pointers renames Buffer.Head;

This Node : Pointers;

-- --begin procedure Retrieve_Element

begin

--make sure an element is available
if Current_Length = -1 then
 raise Buffer_Not_Initialized;

```
elsif Current_Length = 0 then
    raise Buffer_Empty;
end if;
```

```
-- -- save dummy node in the available space list
This_Node := Head;
Head := Head.Next;
Save Node (Saved Node => This Node);
```

- -- -- retrieve element (its node becomes the new dummy node) Old_Element := Head.Data;
- -- --update buffer status
 Current_Length := Current_Length 1;

end Retrieve Element ;

3.3.7.1.9.4.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded_FIFO_Buffer:

	Name		Туре		Description	Ī
	Save_Node	1	procedure	1	Handles placing a node in the available space list	Ī

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Abstract_Data_Structures. Unbounded_FIFO_Buffer package:

Ī	Name		Туре		Des	cription							****	
	Elements	1	private	1	User	defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

Page 1778

	Name	Туре	Range	Description	
	Nodes	record	N/A 	A single entity in the buffer; contains data and a pointer to the next node	
1	Pointers	access	N/A	Points to a node in the buffer	
	Buffers	record 	N/A	Record containing the value of the current length, head, and tail of the buffer	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Unbounded_FIFO_-Buffer:

	Name	Description	Ī
	Buffer_Empty Buffer_Not_ Initialized	Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer Raised if an attempt is made to use an uninitialized buffer	

3.3.7.1.9.4.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name		When/Why	Raised	
Buffer_Em Buffer_No	pty t_Initialized	Raised if buffer Raised if uninitia	an attempt is made to access an empty an attempt is made to use an lized buffer	

3.3.7.1.9.4.10.6 PEEK UNIT DESIGN

This function returns the oldest element in the FIFO buffer.

If the buffer has not been initialized, a Buffer_Not_Initialized exception is raised.

If the buffer is empty, a Buffer Empty exception is raised.

3.3.7.1.9.4.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.3.7.1.9.4.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.4.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	Ī
	Buffer		Buffers		in out		FIFO buffer being accessed	

•

3.3.7.1.9.4.10.6.4 LOCAL DATA

None.

3.3.7.1.9.4.10.6.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.4.10.6.6 PROCESSING

The following describes the processing performed by this part:

function Peek (Buffer : in Buffers) return Elements is

-- --declaration section -- --declaration section -- Current_Length : INTEGER renames Buffer.Current_Length; Head : Pointers renames Buffer.Head; -- --begin function Peek -- --begin function Peek

begin

```
-- --make sure something is there to look at
    if Current Length = -1 then
        raise Buffer_Not Initialized;
    elsif Current Length = 0 then
        raise Buffer_Empty;
    end if;
    return Head.Next.Data;
end Peek ;
```

Page 1780

Q

3.3.7.1.9.4.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Abstract_Data_Structures. Unbounded_FIFO_Buffer package:

1	Name		Туре		Desc	cription								
	Elements		private	Ι	User	defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

	Name	Type	Range	Description	
	Nodes	record	N/A	A single entity in the buffer; contains data and a pointer to the next node	
	Pointers Buffers	access record	N/A N/A	Points to a node in the buffer Record containing the value of the current length, head, and tail of the buffer	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Unbounded_FIFO_-Buffer:

	Name		Description	Ī
	Buffer_Empty Buffer_Not_ Initialized		Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer Raised if an attempt is made to use an uninitialized buffer	

3.3.7.1.9.4.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name		When/Why	Ra	ised							
	Buffer_Empty		Raised if buffer Raised if	an	attempt	is	made	to	access	an	empty	
	burrer_Not_Initialized		uninitial	.iz	ed buffer	:	maue	10	use an			ł

3.3.7.1.9.4.10.7 BUFFER STATUS UNIT DESIGN

This function returns the status of the buffer based on the following algorithm:

if buffer has never been initialized then status is uninitialized elsif buffer has no nodes in it then status is empty else status is available

3.3.7.1.9.4.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.3.7.1.9.4.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.4.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	Туре	Mo	de	Desc	ription	1 1		I
	Buffer	Buffers	in	out	FIFO	buffer	being	accessed	

3.3.7.1.9.4.10.7.4 LOCAL DATA

None.

3.3.7.1.9.4.10.7.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.4.10.7.6 PROCESSING

The following describes the processing performed by this part:

function Buffer Status (Buffer : in Buffers) return Buffer Statuses is

8

Current_Length : INTEGER renames Buffer.Current_Length; Status : Buffer Statuses;

-- --begin function Buffer Status

begin

1.3

ì

.

if Current_Length = -1 then
 Status := Uninitialized;

elsif Current Length = 0 then
 Status := Empty;

else
 Status := Available;

end if;

return Status;

end Buffer Status ;

3.3.7.1.9.4.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of Abstract Data Structures. Unbounded FIFO Buffer:

I	Name		Туре	1	Range		Desc	riŗ	otion				
	Buffer_ Statuses		discrete type		Empty, Available, Uninitialized	U	lsed buff	to er	indicate	the	status	of	the

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

	Name	Ι	Туре	1	Range		Description	Ī
	Buffers		record		N/A		Record containing the value of the current length, head, and tail of the buffer	

3.3.7.1.9.4.10.7.8 LIMITATIONS

None.

3.3.7.1.9.4.10.8 BUFFER_LENGTH UNIT DESIGN

This function returns the length of the current buffer.

If the buffer has not been initialized, a Buffer_Not_Initialized exception is raised.

3.3.7.1.9.4.10.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R164.

3.3.7.1.9.4.10.8.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.4.10.8.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name		Туре		Mode		Description	
	Buffer	1	Buffers	1	in out	1	FIFO buffer being accessed	1

3.3.7.1.9.4.10.8.4 LOCAL DATA

None.

3.3.7.1.9.4.10.8.5 PROCESS CONTROL

Not applicable.

Page 1784

Q

3.3.7.1.9.4.10.8.6 PROCESSING

The following describes the processing performed by this part:

function Buffer Length (Buffer : in Buffers) return NATURAL is

-- --declaration section

Current Length : INTEGER renames Buffer.Current Length;

-- --begin function Buffer_Length

begin

133

;

50

```
-- --make sure the buffer has a length
if Current_Length = -1 then
raise Buffer_Not_Initialized;
end if;
```

return Current Length;

end Buffer_Length ;

3.3.7.1.9.4.10.8.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

	Name	Туре	Range	Description	
	Buffers	record	N/A	Record containing the value of the current length, head, and tail of the buffer	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Unbounded_FIFO_-Buffer:

	Name		Descri	pti	on									
	Buffer_Not_ Initialized		Raised	if	an	attempt	is	made	to	use	an	uninitialized	buffer	

3.3.7.1.9.4.10.8.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why Raised	
	Buffer_Not_Initialized	Raised if an attempt is made to use an uninitialized buffer	

3.3.7.1.9.4.10.9 DOT_NEXT UNIT DESIGN

Given an input pointer P, this function returns the value of P.Next.

3.3.7.1.9.4.10.9.1 REQUIREMENTS ALLOCATION

None.

3.3.7.1.9.4.10.9.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.4.10.9.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	
	Ptr		Pointers		in		Pointer to the node whose "next" entry is to be returned	

3.3.7.1.9.4.10.9.4 LOCAL DATA

None.



R

3.3.7.1.9.4.10.9.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.4.10.9.6 PROCESSING

The following describes the processing performed by this part:

function Dot_Next (Ptr : in Pointers) return Pointers is
begin
 return Ptr.Next;
end Dot_Next;

3.3.7.1.9.4.10.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top-level component:

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

l	Name	Туре	Range	Description	
	Nodes	recor	1 N/A	A single entity in the buffer; contains data and a pointer to the next node	
	Pointers	acces	s N/A	Points to a node in the buffer	İ
	Buffers	record	I N/A	Record containing the value of the curren length, head, and tail of the buffer	t

3.3.7.1.9.4.10.9.8 LIMITATIONS

None.

Ċ

3.3.7.1.9.4.10.10 SET NEXT UNIT DESIGN

Given an two input pointers, P and Q, this procedure sets P.Next equal to Q.

3.3.7.1.9.4.10.10.1 REQUIREMENTS ALLOCATION

None.

Page 1787

3.3.7.1.9.4.10.10.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.4.10.10.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	
	Ptr		Pointers		in		Pointer to the node whose "next" entry is to be modified	
ĺ	Ptr_dot_Next	İ	Pointers	I	in	İ	Value to which Ptr.Next is to be set	İ

3.3.7.1.9.4.10.10.4 LOCAL DATA

None.

3.3.7.1.9.4.10.10.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.4.10.10.6 PROCESSING

The following describes the processing performed by this part:

3.3.7.1.9.4.10.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top-level component:

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

1	Name	Туре	Range	Description
	Nodes	record	N/A	A single entity in the buffer; contains data and a pointer to the next node
	Pointers Buffers	access record	N/A N/A	Points to a node in the buffer Record containing the value of the current length, head, and tail of the buffer

3.3.7.1.9.4.10.10.8 LIMITATIONS

None.

3.3.7.1.9.5 NONBLOCKING CIRCULAR BUFFER PACKAGE DESIGN (CATALOG #P333-0)

This generic package defines the data type and contains the operations required to perform circular buffering operations on incoming data. These operations are performed in a non-blocking fashion such that if the buffer is full, incoming data will overwrite old data. The head of the buffer always points to a dummy node. The first node following the dummy node contains the next piece of data to be retrieved. The tail always points to where the next element should be added. If the tail points to the element immediately in front of the head, the buffer is empty. If the tail points to the same element as the head, the buffer is full. This is illustrated below.

Empty circular buffer: +-+ <-----Head +-+ +-+ <----Tail +-+ +-+ +-+

Full circular buffer: Tail---->+-+ <-----Head +-+ +-+ +-+ +-+ +-+ +-+ +-+

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.9.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R126.

3.3.7.1.9.5.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of the Abstract_Data_Structures package:

Data types:

1.3

The following table summarizes the generic formal types required by this part:

	Name		Туре		Desc	cription								
1	Elements		private		User	defined	type	of	data	contained	in	the	buffer	

Data objects:

The following table summarizes the generic formal objects required by this part:

	Name		Туре		Value		Description	
	Initial_ Buffer_Size		POSITIVE		N/A		Maximum number of elements which can be in the buffer at any given time	

3.3.7.1.9.5.4 LOCAL DATA

None.

3.3.7.1.9.5.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.5.6 PROCESSING

The follow. g describes the processing performed by this part:

separate (Abstract_Data_Structures)
package body Nonblocking Circular Buffer is

end Nonblocking_Circular_Buffer;

3.3.7.1.9.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following data types were previously defined in this part's package specification:

Page 1790

0

	Name	Туре	Range	Description	
	Buffer_Range Buffer_ Statuses	NATURAL subtype discrete type	0 Buffer_Size Empty, Available, Full	Used to dimension the list of elements Used to indicate the status of the buffer	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Nonblocking_Circular_Buffer package:

1	Name	Туре	Range	Description	
	Lists Buffers	array record 	N/A N/A 	Array of elements List of data along with relevant information	

Data objects:

The following data objects were previously defined in this part's package specification:

	Name		Туре		Value		Description	1
	Buffer_Size		POSITIVE		Initial_ Buffer_Size		Number of usable elements in a buffer	

Exceptions:

The following exceptions were previously defined in this part's package specification:

Ī	Name	Description	
	Buffer_Empty	Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer	

3.3.7.1.9.5.8 LIMITATIONS

None.

3.3.7.1.9.5.9 LLCSC DESIGN

None.

3.3.7.1.9.5.10 UNIT DESIGN

3.3.7.1.9.5.10.1 CLEAR BUFFER UNIT DESIGN

This procedure clears a buffer by setting the Head to 0, the Tail to 1, and the length to 0.

3.3.7.1.9.5.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R126.

3.3.7.1.9.5.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.5.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name		Туре	1	Mode		Description	
	Buffer		Buffers		out		Nonblocking circular buffer being accessed	

3.3.7.1.9.5.10.1.4 LOCAL DATA

None.

3.3.7.1.9.5.10.1.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.5.10.1.6 PRCCESSING

The following describes the processing performed by this part:

procedure Clear Buffer (Buffer : out Buffers) is

-- -- declaration section

-- ----

Head : Buffer_Range renames Buffer.Head; Tail : Buffer_Range renames Buffer.Tail; Current_Length : Buffer_Range renames Buffer.Current_Length;

-- ----

X

```
-- -- begin procedure Clear_Buffer
```

begin

```
Head := 0;
Tail := 1;
Current_Length := 0;
```

end Clear Buffer ;

3.3.7.1.9.5.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of the Nonblocking_Circular_Buffer package:

1	Name		Туре	1	Range		Description	1
	Buffer_Range		NATURAL subtype		0 Buffer_Size		Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Nonblocking_Circular_Buffer package:

Ī	Name	Туре	Range	Description	
	Buffers	record	N/A	List of data along with relevant information	

3.3.7.1.9.5.10.1.8 LIMITATIONS

None.

3.3.7.1.9.5.10.2 ADD ELEMENT UNIT DESIGN

This procedure adds an element to the end of the buffer, overwriting old data if the buffer is full. If data was overwritten, both the head and tail of the buffer are adjusted to reflect the current status of the buffer. If data was not overwritten, only the tail of the buffer is adjusted.

3.3.7.1.9.5.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R126.

3.3.7.1.9.5.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.5.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	
	Buffer New_Element		Buffers Elements		out in		Circular buffer being accessed Element to be added to the buffer	

3.3.7.1.9.5.10.2.4 LOCAL DATA

None.

3.3.7.1.9.5.10.2.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.5.10.2.6 PROCESSING

The following describes the processing performed by this part:

procedure Add_Element (New Element : in Elements; Buffer : in out Buffers) is

-- -- declaration section

> Head : Buffer_Range renames Buffer.Head; Tail : Buffer_Range renames Buffer.Tail; Current_Length : Buffer_Range renames Buffer.Current_Length; List : Lists renames Buffer.List;

-- --begin procedure Add_Element

begin

List(Tail) := New Element;

64

```
if Head = Tail then
         --buffer was already full and an element was overwritten; therefore,
_ _
         -- both head and tail need to be advanced, but Current Length does
--
         --not need to be changed
         if Tail = Buffer Size then
            Head := 0;
            Tail := 0;
         else
            Head := Head + 1;
            Tail := Tail + 1;
         end if:
     else
         --buffer was not already full; therefore, the Current Length needs
         -- to be increment and only the tail needs to be advanced
         if Tail = Buffer Size then
            Tail := 0;
        else
            Tail := Tail + 1;
        end if;
        Current Length := Current Length + 1;
     end if;
  end Add Element ;
```

3.3.7.1.9.5.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

.

The following table summarizes the types required by this part and defined as generic formal types to the Nonblocking Circular Buffer package:

	Name		Туре		Description								
	Elements		private	1	User defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the package specification of the Nonblocking_Circular_Buffer package:

Page 1795

Page 1796

	Name		Туре		Range		Description	
	Buffer_Range		NATURAL subtype		0 Buffer_Size		Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Nonblocking_Circular_Buffer package:

	Name	1	Туре		Range		Description	
	Buffers		record		N/A		List of data along with relevant information	
İ	Lists	İ	array	i	N/A		Array of elements	i

Data objects:

The following table summarizes the types required by this part and defined in the package specification of Nonblocking_Circular_Buffer:

Ī	Name	ame Type		Description	
	Buffer_Size	POSITIVE	Initial_ Buffer_Size	Number of usable elements in a buffer	

3.3.7.1.9.5.10.2.8 LIMITATIONS

None.

3.3.7.1.9.5.10.3 RETRIEVE ELEMENT UNIT DESIGN

This procedure returns the first element in the circular buffer.

If there are no elements in the buffer, a Buffer Empty exception is raised.

3.3.7.1.9.5.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R126.

3.3.7.1.9.5.10.3.2 LOCAL ENTITIES DESIGN

None.
3.3.7.1.9.5.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode	1	Description	
	Buffer Old_Element		Buffers Elements		out out		Circular buffer being accessed Element retrieved from the buffer	

3.3.7.1.9.5.10.3.4 LOCAL DATA

None.

--

3.3.7.1.9.5.10.3.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.5.10.3.6 PROCESSING

The following describes the processing performed by this part:

procedure Retrieve_Element (Buffer : in out Buffers; Old Element : out Elements) is

-- -- declaration section

Head : Buffer_Range renames Buffer.Head; Tail : Buffer_Range renames Buffer.Tail; Current_Length : Buffer_Range renames Buffer.Current_Length; List : Lists renames Buffer.List;

---begin procedure Retrieve_Element

begin

```
-- --make sure there is something there to retrieve
if Current_Length = 0 then
raise Buffer_Empty;
end if;
-- --advance the head to get to the next element to go out
if Head = Buffer_Size then
Head := 0;
else
Head := Head + 1;
end if;
```

Page 1798

Ś

-- --now retrieve the element and update the state of the buffer Old_Element := List(Head); Current Length := Current Length - 1;

end Retrieve_Element ;

3.3.7.1.9.5.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal types to the Nonblocking_Circular_Buffer package:

	Name		Туре		Dese	cription								
	Elements	1	private		User	defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the package specification of the Nonblocking Circular Buffer package:

	Name	Туре	Range	Description	1
	Buffer_Range	NATURAL subtype	0 Buffer_Size	Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Nonblocking_Circular_Buffer package:

	Name		Туре		Range		Description	
	Buffers		record		N/A		List of data along with relevant	
ļ	Lists	I	array	İ	N/A		Array of elements	ļ

Data objects:

The following table summarizes the types required by this part and defined in the package specification of Nonblocking Circular Buffer:

	Name		Туре	1	Value		Description	Ī
	Buffer_Size		POSITIVE		Initial_ Buffer_Size		Number of usable elements in a buffer	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Nonblocking Circular Buffer:

1	Name		Description	
	Buffer_Empty	 	Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer	

3.3.7.1.9.5.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

 Name
 Description

 Buffer_Empty
 Error condition raised if an attempt is made to look at or

 retrieve elements from an empty buffer

3.3.7.1.9.5.10.4 PEEK UNIT DESIGN

This function returns the data contained in the first element in the buffer without changing the state of the buffer (i.e., the element is not removed from the buffer).

If there are no elements in the buffer, a Buffer Empty exception is raised.

3.3.7.1.9.5.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R126.

3.3.7.1.9.5.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.5.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Mode	Description	I
	Buffer	Buffers	out	Circular buffer being accessed	I

3.3.7.1.9.5.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

	Name		Туре		Va	alue		Description	
	Spot	1	Buffer_	Range	N//	A	١	Marks the spot in the buffer containing the element to be looked at	

3.3.7.1.9.5.10.4.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.5.10.4.6 PROCESSING

The following describes the processing performed by this part:

function Peek (Buffer : in Buffers) return Elements is

-- --declaration section

Head : Buffer_Range renames Buffer.Head; Current_Length : Buffer_Range renames Buffer.Current_Length; List : Lists renames Buffer.List;

Spot : Buffer Range;

-- -- begin function Peek

__ ____

begin

```
-- --make sure there is something to peek at
if Current_Length = 0 then
raise Buffer_Empty;
end if;
```

-- -- determine location of desired element if Head = Buffer Size then Page 1800



```
Spot := 0;
else
   Spot := Head + 1;
end if;
```

```
--return requested element
return List(Spot);
```

end Peek ;

3.3.7.1.9.5.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal types to the Nonblocking Circular_Buffer package:

	Name		Туре		Desc	cription								
	Elements	1	private		User	defined	type	of	data	contained	in	the	buffer	

The following table summarizes the types required by this part and defined in the package specification of the Nonblocking_Circular_Buffer package:

	Name		Туре	1	Range		Description	I
	Buffer_Range		NATURAL subtype		0 Buffer_Size		Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract Data Structures.Nonblocking Circular Buffer package:

1	Name	Туре	Range	Description	
	Buffers	record	N/A	List of data along with relevant information	
1	Lists	array	N/A	Array of elements	

Data objects:

The following table summarize the types required by this part and defined in the package specification of Nonblocking Circular Buffer:

_ _

Page 1802

	Name	Туре	Value	Description	-
	Buffer_Size 	POSITIVE	Initial_ Buffer_Size	Number of usable elements in a buffer	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Nonblocking_Circular_Buffer:

1	Name		Description	
	Buffer_Empty		Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer	

3.3.7.1.9.5.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name		Description	
	Buffer_Empty		Error condition raised if an attempt is made to look at or retrieve elements from an empty buffer	

3.3.7.1.9.5.10.5 BUFFER STATUS UNIT DESIGN

This function returns the current status of the buffer according to the following algorithm:

if there are no elements in the buffer then buffer status is empty elsif if the buffer contains the maximum number of elements buffer status is full else buffer status is available end if;

3.3.7.1.9.5.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R126.

3.3.7.1.9.5.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.5.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре	Mode	Description	
Buffer	Buffers	out	Circular buffer being accessed	

3.3.7.1.9.5.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

I	Name		Туре		Value	1	Description	Ī
	Status		Buffer_Statuses	I	N/A		Current status of the buffer	

3.3.7.1.9.5.10.5.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.5.10.5.6 PROCESSING

The following describes the processing performed by this part:

function Buffer Status (Buffer : in Buffers) return Buffer Statuses is

-- -- declaration section

Current Length : Buffer Range renames Buffer.Current Length;

Status : Buffer_Statuses;

-- --begin function Buffer_Status

begin

```
if Current_Length = 0 then
    Status := Empty;
elsif Current_Length = Buffer_Size then
    Status := Full;
else
    Status := Available;
```

Page 1803

Page 1804

end if;

return Status;

end Buffer_Status ;

3.3.7.1.9.5.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of the Nonblocking_Circular_Buffer package:

	Name	Туре	Range	Description	1
	Buffer_Range Buffer_ Statuses	NATURAL subtype discrete type	0 Buffer_Size Empty, Available, Full	Used to dimension the list of elements Used to indicate the status of the buffer	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Nonblocking_Circular_Buffer package:

	Name		Туре		Range	1	Description	
	Buffers		record		N/A		List of data along with relevant information	

Data objects:

The following table summarizes the types required by this part and defined in the package specification of Nonblocking Circular Buffer:

	Name	Туре	Value	Description	
	Buffer_Size 	POSITIVE	Initial_ Buffer_Size	Number of usable elements in a buffer	

3.3.7.1.9.5.10.5.8 LIMITATIONS

None.

3.3.7.1.9.5.10.6 BUFFER_LENGTH UNIT DESIGN This function returns the current length of the buffer.

3.3.7.1.9.5.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R126.

3.3.7.1.9.5.10.6.2 LOCAL ENTITIES DESIGN None.

3.3.7.1.9.5.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode	1	Description	Ī
	Buffer	1	Buffers		out	1	Circular buffer being accessed	1

```
3.3.7.1.9.5.10.6.4 LOCAL DATA
```

None.

Ö

19

3.3.7.1.9.5.10.6.5 PROCESS CONTROL Not applicable.

3.3.7.1.9.5.10.6.6 PROCESSING

The following describes the processing performed by this part:

function Buffer_Length (Buffer : in Buffers) return Buffer_Range is
begin

return Buffer.Current_Length;

end Buffer_Length ;

3.3.7.1.9.5.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of the Nonblocking Circular Buffer package:

1	Name		Туре		Range		Description	1
	Buffer_Range		NATURAL subtype	 	0 Buffer_Size		Used to dimension the list of elements	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Nonblocking_Circular_Buffer package:

	Name		Туре		Range		Description	
	Buffers		record		N/A		List of data along with relevant information	

3.3.7.1.9.5.10.6.8 LIMITATIONS

None.

3.3.7.1.9.6 UNBOUNDED PRIORITY QUEUE PACKAGE DESIGN (CATALOG #P334-0)

This generic package defines the data type and contains the operations required to perform priority queueing operations on incoming data. The head of the queue always points to a dummy node. The node following the dummy node contains the element with the highest priority. The tail always points to the element with the lowest priority.

The elements will be ordered in the queue such that: 1) Elements with higher priorities are placed before those with lower priorities. 2) Elements with the same priority are arranged in the queue in a first-in-first-out manner.

A queue must be initialized before it is used. If an attempt is made to use an uninitialized queue, the exception Queue Not Initialized will be raised. The Initialized Queue procedure returns an initialized queue. The Clear Queue procedure returns the nodes of a queue to the available space list and then returns an initialized queue.

An available space list is maintained local to this part. When this part is elaborated the available space list will have a dummy node plus Initial -Available Space Size nodes. When nodes are added to the queue, the Add Element routine will try to get a node from the available space list before attempting to allocate more memory. When the Retrieve Element routine is called, the unused node will be returned to the available space list for later use. The memory committed to the available space may be deallocated by calling the Free Memory procedure.

Page 1806

ব্

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.9.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.2 LOCAL ENTITIES DESIGN

Data structures:

An available space list is maintain local to this part's package body.

Subprograms:

The following subprograms are contained local to this body:

Name	1	Туре		Description	
Free Node Dot_Next		proc edure function		Instantiation of UNCHECKED DEALLOCATION Given a pointer P, this function returns the value of P.Next	
Set_Next		procedure		Given two points P & Q, this procedure sets P.Next = Q	

The following subprograms are contained in this part as a result of renaming operations on identically named routines contained in the locally instantiated Available Space Operations package.

	Name	Туре	Description	
	New_Node Save_Node Save_List	function procedure procedure	Returns a node to the calling routine; will get a node from the available space list if possible, otherwise will allocate a new node Handles placing a node in the available space list Handles placing a list of nodes in the available space list	

This package body contains code to initialize the Available Space List. This code is executed when the package is elaborated. If the generic formal object Initial Available Space Size is greater than or equal to 1, this routine then places the requested number of nodes (in addition to the dummy node) in the available space list.

3.3.7.1.9.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:



6

Page 1807

The following generic parameters were previously defined when this part was specified in the package specification of the Abstract_Data_Structures package:

Data types:

The following table summarizes the generic formal types required by this part:

	Name	1	Туре		Description	
	Elements Priorities		private private		User defined type of data contained in the queue User defined type determining the priority of the node	

Data objects:

The following table summarizes the generic formal objects required by this part:

	Name		Туре		Description	
	Initial Available_ Space_Size		NATURAL		Number of available nodes to be initially placed in the available space list	

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

1	Name		Туре	1	Description	
	">"		function		Used to determine ordering of priorities	

3.3.7.1.9.6.4 LOCAL DATA

Data objects:

The following table summarizes the data objects defined by this part as the result of renames:

	Name	1	Туре		Value		Description	1
	Available_ Length Available_ Head		INTEGER Pointers		Available_Space. Current_Length Available_Space. Head		Indicates the current length of the available space list Points to the head node in the available space list	
	Available_ Tail		Pointers		Available_Space. Tail		Points to the tail node in the available space list	

.

3.3.7.1.9.6.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.6.6 PROCESSING

The following describes the processing performed by this part:

with UNCHECKED_DEALLOCATION; separate (Abstract_Data Structures) package body Unbounded_Priority_Queue is

-- -- this pointers is accessed ONLY when setting up the Available_Space Initial_Head : Pointers := new Nodes;

Available_Space : Queues := (Current_Length => 0, Head => Initial_Head, Tail => Initial_Head);

Available Length : INTEGER renames Available Space.Current Length; Available Head : Pointers renames Available Space.Head; Available Tail : Pointers renames Available Space.Tail;

function Dot Next (Ptr : in Pointers) return Pointers;

package Available Space Operations is new Available Space List Operations (Nodes => Nodes, Pointers => Pointers, Available Length => Available Length, Available Head => Available Head, Available Tail => Available Tail);

function New_Node return Pointers
 renames Available_Space_Operations.New_Node;

procedure Save_List (Saved_Head : in Pointers; Saved_Tail : in Pointers; Node Count : in POSITIVE)



renames Available_Space_Operations.Save_List;

```
--begin package Unbounded Priority Queue
--(see header for package body for details)
begin
-- -- set up available space list if one is desired
  if Initial Available Space Size > 0 then
     Add Nodes To Available Space List:
       for I in I..Initial Available Space Size loop
          Available_Tail.Next := NEW Nodes;
          Available Tail := Available Tail.Next;
       end loop Add Nodes to Available Space List;
     Available Length := Initial Available Space Size;
  end if;
end Unbounded Priority Queue;
3.3.7.1.9.6.7 UTILIZATION OF OTHER ELEMENTS
The following library units are with'd by this part:
   1. Unchecked Deallocation
Subprograms and task entries:
The following table describes the subroutines required by this part:
                                          | Name | Type | Source | Description
 _____
           | Unchecked____| generic | N/A | Used to deallocate memory
| Deallocation | function | |
Exceptions:
The following table describes the exceptions required by this part and defined
in the Ada predefined package STANDARD:
```

1	Name		Description	
	Storage_Error		Raised when an attempt is made to dynamically allocate more memory than is available	

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

ę

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table describes the packages required by this part and specified in the package body of the Abstract Data Structures package:

	Name		Туре	Ι	Description	
	Available_Space_ List_Operations		generic package		Contains the routines required to retrieve a node from and place a node in the available space list	

Data types:

The following data types were previously defined in this part's package specification:

	Name	Туре	Range	Description	
	Queue Statuses	discrete type	Empty, Available, Uninitialized	Used to indicate the status of queue 	the

The following data types were previously defined in the private portion of this part's package specification:

	Name	Туре	Range	Description	
	Nodes	record	N/A	A single entity in the queue; contains data and a pointer to the next node	
	Pointers Queues	access record	N/A N/A 	Points to a node in the queue Record containing the value of the current length, head, and tail of the queue	

Exceptions:

The following exceptions were previously defined in this part's package specification:

 Name
 Description

 Queue_Empty
 Error condition raised if an attempt is made to look at or

 retrieve elements from an empty queue

 Queue_Not

 Indicates an attempt was made to use an uninitialized queue

3.3.7.1.9.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why Raised	
	Storage_Error	Raised during elaboration of this package if an attempt is made to allocate memory when no more is available	

3.3.7.1.9.6.9 LLCSC DESIGN

None.

3.3.7.1.9.6.10 UNIT DESIGN

3.3.7.1.9.6.10.1 INITIALIZE UNIT DESIGN

This procedure initializes a queue by placing a dummy node in it, pointing the head and the tail to the dummy node, and setting the length to O.

3.3.7.1.9.6.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Mode	Description	
	Queue	Queues 	in out	Unbounded priority queue being manipulated	

3.3.7.1.9.6.10.1.4 LOCAL DATA

None.

Å

```
3.3.7.1.9.6.10.1.5 PROCESS CONTROL
Not applicable.
```

3.3.7.1.9.6.10.1.6 PROCESSING

The following describes the processing performed by this part:

procedure Initialize (Queue : in out Queues) is

_____ --declaration section ---Current Length : INTEGER renames Queue.Current Length; : Pointers renames Queue.Head; Head Tail : Pointers renames Queue.Tail; -- -------- --begin procedure Initialize begin if Current Length = -1 then --handle an uninitialized queue ___ Head := New Node: Tail := Head; Current Length := 0; elsif Current_Length > 0 then --handle a queue that has something in it ___ Clear Queue(Queue => Queue); else --current length = 0 so it is already initialized ----NULL; end if; end Initialize ; 3.3.7.1.9.6.10.1.7 UTILIZATION OF OTHER ELEMENTS UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT: The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package specification of Unbounded Priority Queue:

	Name		Туре	1	Description	
	Clear_ Queue		procedure		Returns all the nodes in a queue to the available space list	

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded_Priority_Queue:

	Name		Туре		Description	
	New_Node		function		Returns a node to the calling routine; will get a node from the available space list if possible, otherwise will allocate a new node	

Data types:

The following table describes the data types required by this part and defined in the private portion of the Abstract_Data_Structures.Unbounded_Priority_Queue package:

	Name	Туре	Range	Description	
	Nodes Pointers Queues	record access record	N/A N/A N/A	A single entity in the queue; contains data and a pointer to the next node Points to a node in the queue Record containing the value of the current length, head, and tail of the queue	

3.3.7.1.9.6.10.1.8 LIMITATIONS

None.

3.3.7.1.9.6.10.2 CLEAR QUEUE UNIT DESIGN

This procedure removes the nodes from a queue and places them in an available space list.

The Queue Not Initialized exception is raised if this routine is called with an uninitialized queue.

3.3.7.1.9.6.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Mode	Description	
	Queue	Queues	in out	Unbounded priority queue being manipulated	

3.3.7.1.9.6.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained local to this part:

	Name		Туре	I	Value		Description	
	This_Node		Pointers		N/A		Points to the node to be returned to the available space list	

3.3.7.1.9.6.10.2.5 PROCESS CONTROL

Not applicable

3.3.7.1.9.6.10.2.6 PROCESSING

The following (Fires) ibes the processing performed by this part:

procedure that we (Queue : in out Queues) is

begin

```
--make sure this is an initialized queue
      if Current Length = -1 then
         raise Queue Not Initialized;
      elsif Current Length > 0 then
        --placed nodes in the available space list
        Save_List (Saved_Head => Head.Next,
Saved_Tail => Tail,
                  Node Count => Current Length);
        --reinitialize queue variables
        Current Length := 0;
        Head.Next := NULL;
        Tail
                     := Head:
      end if;
   end Clear Queue ;
3.3.7.1.9.6.10.2.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:
The following tables describe the elements used by this part but defined
elsewhere in the parent top level component:
Subprograms and task entries:
The following table summarizes the subroutines and task entries required by
this part and defined in the package body of Unbounded Priority Queue:
 Save List | procedure | Handles placing a list of nodes in the available
   space list
                     _____
Data types:
The following table describes the data types required by this part and defined
in the private portion of the Abstract Data Structures. Unbounded Priority Queue
package:
```

	Name	Туре	Range	Description	
	Nodes	record	N/A	A single entity in the queue; contains data and a pointer to the next node	
İ İ	Pointers Queues	access record	N/A N/A	Points to a node in the queue Record containing the value of the current length, head, and tail of the queue	

Page 1816

Exceptions:

The following table summarizes the exceptions required by this part and defined elsewhere in the package specification of Abstract_Data_Structures.Unbounded_- Priority Queue:

Name	Description	
Queue_Empty 	Error condition raised if an attempt is made to look at or retrieve elements from an empty queue	r

3.3.7.1.9.6.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why	Raised			
	Queue_Not_Initialized	Raised if uninitia	an attempt lized queue	is made	to manipulate an	

3.3.7.1.9.6.10.3 FREE MEMORY UNIT DESIGN

This procedure deallocates the memory taken up by the available space list.

3.3.7.1.9.6.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.10.3.2 LOCAL ENTITIES DESIGN

None.

Ö

3.3.7.1.9.6.10.3.3 INPUT/OUTPUT

None.

3.3.7.1.9.6.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

	Name		Туре		Value		Description	-
	Node_to_be_Freed		Pointers		N/A		Points to the node to be deallocated	

3.3.7.1.9.6.10.3.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.6.10.3.6 PROCESSING

The following describes the processing performed by this part:

procedure Free_Memory is

 declaration	section

Node_to_be_Freed : Pointers;

-- --begin procedure Free_Memory

begin

```
Clear Out Available Space List:

while Available Head /= Available Tail loop

Node To Be Freed := Available Head;

Available Head := Available Head.Next;

Free Node (Which Node => Node to be Freed);

end loop Clear Out Available Space List;
```

Available Length := 0;

end Free Memory ;

3.3.7.1.9.6.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded_Priority_Queue:

Page 1818

	Name		Туре		Description	
	Free_Node		procedure		Instantiation of UNCHECKED_DEALLOCATION	Ī

Data types:

The following table describes the data types required by this part and defined in the private portion of the Abstract_Data_Structures.Unbounded_Priority_Queue package:

	Name	Туре	Range	Description
	Nodes Pointers	record access	N/A N/A	A single entity in the queue; contains data and a pointer to the next node Points to a node in the queue
İ	Queues	record	N/A	Record containing the value of the current length, head, and tail of the queue

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Abstract Data Structures. Unbounded Priority Queue:

-	Name		Туре		Value		Description	-
-	Availahla	 	TNTECER	 	Available Space	 1	Indicates the ourrent length of	÷
1	Length		INIEGER		Current_Length		the available space list	
	Available_ Head		Pointers		Available_Space.		Points to the head node in the available snace list	
	Available_	ļ	Pointers	l	Available_Space.		Points to the tail node in the	ļ
ł	Tail	ļ		I	Tail		available space list	ļ

3.3.7.1.9.6.10.3.8 LIMITATIONS

None.

3.3.7.1.9.6.10.4 ADD ELEMENT UNIT DESIGN

This procedure adds an element to the queue. The elements are added such that the new element is added before the first element which has a smaller priority and after all other elements which a greater or equal priority.

The Queue_Empty exception is raised if this routine is called with an empty queue.

The Queue_Not_Initialized exception is raised if this routine is called with an uninitialized queue.

The Storage Error exception is raised if a call to this routine requires memory to be dynamically allocated when no more memory is available.

3.3.7.1.9.6.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name		Туре		Mode		Description	I
	New_Element New_Priority Queue		Elements Prioritie Queues	s 	in in in out		Element to be placed in the queue Priority of the element to be placed in the queue Unbounded priority queue being manipulated	

3.3.7.1.9.6.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

	Name		Туре		Value		Description	
	Before		Pointers		N/A		Points to the element which will go before the new element	
	Here		Pointers		N/A		Points to the node to be added to the queue	

3.3.7.1.9.6.10.4.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.6.10.4.6 PROCESSING

The following describes the processing performed by this part:

procedure Add_Element (New_Element : in Elements; New_Priority : in Priorities; ≥ 1820



.

```
Queue : in out Queues) is
---
     --declaration section
--
      _____
--
     Current Length : INTEGER renames Queue.Current Length;
     Head : Pointers renames Queue.Head;
Tail : Pointers renames Queue.Tail;
     Before : Pointers;
Here : Pointers;
-- -- begin procedure Add Element
begin
     --make sure queue has been initialized
---
     if Current Length = -1 then
        raise Queue Not Initialized;
     end if;
     --find the nodes which are to go before and after the new element
--
     Before := Head;
     loop
        exit when (Before = Tail) or else
                  (New_Priority > Before.Next.Priority);
        Before := Before.Next;
     end loop;
     -- now get a new node
--
     Here := New Node;
     --set up the new node
---
     Here.Priority := New_Priority;
                                              1
     Here.Data := New Element;
Here.Next := Before.Next;
     Before.Next := Here;
     --readjust the tail, if required
     if Before = Tail then
       Tail := Here;
     end if;
     -- now adjust the queue
--
     Current_Length := Current_Length + 1;
  end Add Element ;
3.3.7.1.9.6.10.4.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:
```

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded Priority Queue:

1	Name		Туре		Description	
	New_Node		function		Returns a node to the calling routine; will get a node from the available space list if possible, otherwise will allocate a new node	

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded_Priority_Queue:

	Name		Туре		Description			Ī
	Free_Node		procedure		Instantiation of	UNCHECKED	DEALLOCATION	1

The following table describes the subroutines required by this part and defined as generic formal subroutines to the Abstract_Data_Structures.Unbounded_-Priority Queue package:

	Name		Туре		Description	
Ī	">"		function		Used to determine ordering of priorities	

Data types:

The following table describes the data types required by this part and defined in the private portion of the Abstract_Data_Structures.Unbounded_Priority_Queue package:

	Name	Туре	Range	Description	1
	Nodes	record	N/A	A single entity in the queue; contains data and a pointer to the next node	
	Pointers Queues	access record	N/A N/A 	Points to a node in the queue Record containing the value of the current length, head, and tail of the queue	

Exceptions:

The following table summarizes the exceptions required by this part and defined elsewhere in the package specification of Abstract_Data_Structures.Unbounded_- Priority_Queue:

Page 1822

Page	1823
------	------

	Name	Description	
	Storage_ Error Queue_Not_ Initialized	Raised when an attempt is made to dynamically allocate more memory than is available Indicates an attempt was made to use an uninitialized queue	

3.3.7.1.9.6.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised	
Storage_Error Queue_Not_Initialized	Raised if an attempt is raised to allocate memory when no more is available Raised if an attempt is made to manipulate an uninitialized queue	,

3.3.7.1.9.6.10.5 RETRIEVE_ELEMENT UNIT DESIGN

This procedure returns the first element in the queue.

The Queue_Empty exception is raised if this routine is called with an empty queue.

The Queue Not Initialized exception is raised if this routine is called with an uninitialized queue.

3.3.7.1.9.6.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Mode Description	
 	Queue	Queues	in out Unbounded priority queue being manipulated	
1	Old_Element	Elements	out Data retrieved from the queue	j

3.3.7.1.9.6.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

	Name		Туре	1	Value		Description	Ī
	This_Node		Pointers		N/A		Points to the node to be returned to the available space list	

3.3.7.1.9.6.10.5.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.6.10.5.6 PROCESSING

The following describes the processing performed by this part:

procedure Retrieve_Element (Queue : in out Queues; Old Element : out Elements) is

-- -- declaration section

> Current_Length : INTEGER renames Queue.Current_Length; Head : Pointers renames Queue.Head;

This_Node : Pointers;

```
-- --begin procedure Retrieve_Element
```

begin

```
-- --make sure an element is available
if Current_Length = -1 then
    raise Queue_Not_Initialized;
elsif Current_Length = 0 then
    raise Queue_Empty;
end if;
```

Page 1824

--save dummy node in the available space list This Node := Head; Head := Head.Next; Save Node (Saved Node => This_Node);

--retrieve element (its node becomes the new dummy node) Old Element := Head.Data;

```
- --update queue status
Current_Length := Current_Length - 1;
```

end Retrieve Element ;

3.3.7.1.9.6.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Unbounded Priority Queue:

Ī		Name		Туре		Description	• ••• •							
Ī	S	Save_Node		procedure		Handles placing	a	node	in	the	available	space	list	ļ

Data types:

The following table describes the data types required by this part and defined in the private portion of the Abstract_Data_Structures.Unbounded_Priority_Queue package:

1	Name	'	Гуре		Range		Description	Ī
	Nodes	r	ecord		N/A		A single entity in the queue; contains data and a pointer to the next node	
	Pointers Queues	a re	ccess ecord		N/A N/A	Ì	Points to a node in the queue Record containing the value of the current length, head, and tail of the queue	

Exceptions:

The following table summarizes the exceptions required by this part and defined elsewhere in the package specification of Abstract_Data_Structures.Unbounded_-Priority_Queue:

Page 1825

	Name	Description	
	Queue_Empty Queue_Not_ Initialized	Error condition raised if an attempt is made to look at or retrieve elements from an empty queue Indicates an attempt was made to use an uninitialized queue	

3.3.7.1.9.6.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

I	Name	When/Wh	/ Raised	
	Queue_Empty Queue_Not_Initialized	Raised i retriev Raised i uninitia	f an attempt is made to look at or from an empty queue an attempt is made to manipulate an alized queue	

3.3.7.1.9.6.10.6 PEEK UNIT DESIGN

This function returns the value of the first element in the queue, but does not change the state of the queue (i.e., the node is not actually removed from the queue).

The Queue_Empty exception is raised if this routine is called with an empty queue.

The Queue Not Initialized exception is raised if this routine is called with an uninitialized queue.

3.3.7.1.9.6.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре	Mode Description	
Queue	Queues	in out Unbounded priority queue being	

3.3.7.1.9.6.10.6.4 LOCAL DATA

None.

3.3.7.1.9.6.10.6.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.6.10.6.6 PROCESSING

The following describes the processing performed by this part:

function Peek (Queue: in Queues) return Elements is

---___ --declaration section -----------

> Current Length : INTEGER renames Queue.Current Length; : Pointers renames Queue.Head; Head

```
-- -- begin function Peek
```

begin

.

--make sure something is there to look at ___ if Current Length = -1 then raise Queue Not Initialized; elsif Current Length = 0 then raise Queue Empty; end if; return Head.Next.Data;

end Peek ;

3.3.7.1.9.6.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table describes the data types required by this part and defined in the private portion of the Abstract_Data_Structures.Unbounded_Priority_Queue package:

1	Name	Туре	Range		Description	
	Nodes	record	N/A		A single entity in the queue; contains data and a pointer to the next node	
İ	Pointers	access	N/A	İ	Points to a node in the queue	Ì
İ	Queues	record	N/A		Record containing the value of the current length, head, and tail of the queue	

Exceptions:

The following table summarizes the exceptions required by this part and defined elsewhere in the package specification of Abstract_Data_Structures.Unbounded_- Priority_Queue:

Name	Description	Ī
Queue_Empty Queue_Not_ Initialized	Error condition raised if an attempt is made to look at or retrieve elements from an empty queue Indicates an attempt was made to use an uninitialized queue	

3.3.7.1.9.6.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why	Raised
	Queue_Empty Queue Not Initialized	Raised if retrieve Raised if	an attempt is made to look at or from an empty queue an attempt is made to manipulate an
 		uninitia	lized queue

3.3.7.1.9.6.10.7 QUEUE_STATUS UNIT DESIGN

This function returns the status of the queue based on the following algorithm:

if the queue has not been initialized then queue status is uninitialized elsif no elements are in the queue then queue status is empty else queue status is available end if;

3.3.7.1.9.6.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	
	Queue)ueues 	i	n out		Unbounded priority queue being manipulated	

3.3.7.1.9.6.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Ī	Name		Туре		Value		Description	
	Status		Queue_Statuses		N/A		Status of the queue	Ī

3.3.7.1.9.6.10.7.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.6.10.7.6 PROCESSING

The following describes the processing performed by this part:

function Queue Status (Queue : in Queues) return Queue Statuses is

-- --declaration section

Current_Length : INTEGER renames Queue.Current_Length; Status : Queue_Statuses;

330

-- -- begin function Queue_Status

- ------

begin

```
if Current_Length = -1 then
    Status := Uninitialized;
elsif Current Length = 0 then
    Status := Empty;
else
    Status := Available;
end if;
return Status;
end Queue_Status ;
```

3.3.7.1.9.6.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table describes the data types required by this part and defined in the package specification of Abstract_Data_Structures.Unbounded_Priority_-Queue:

1	Name		Туре		Range		Des	cri	ption					
	Queue_ Statuses		discrete type		Empty, Available, Uninitialized	ι	Jsed que	to ue	indicate	the	status	of	the	

3.3.7.1.9.6.10.7.8 LIMITATIONS

None.

3.3.7.1.9.6.10.8 QUEUE LENGTH UNIT DESIGN

This function returns the length of a queue.

The Queue Not_Initialized exception is raised if this routine is called with an uninitialized queue.

3.3.7.1.9.6.10.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R165.

3.3.7.1.9.6.10.8.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.8.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name		Туре		Mode		Description	1
	Queue		Queues		in		Unbounded priority queue being manipulated	

3.3.7.1.9.6.10.8.4 LOCAL DATA

None.

3.3.7.1.9.6.10.8.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.6.10.8.6 PROCESSING

The following describes the processing performed by this part:

function Queue Length (Queue : in Queues) return NATURAL is

-- -- declaration section

Current_Length : INTEGER renames Queue.Current_Length;

----begin function Queue_Length

begin

.

```
-- --make sure the queue has a length
if Current_Length = -1 then
raise Queue_Not_Initialized;
end if;
return Current_Length;
end Queue Length;
```

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table describes the data types required by this part and defined in the private portion of the Abstract_Data_Structures.Unbounded_Priority_Queue package:

1	Name		Туре	1	Range		Description	•
	Nodes Pointers Queues		record access record		N/A N/A N/A		A single entity in the queue; contains data and a pointer to the next node Points to a node in the queue Record containing the value of the current	
İ		İ		ļ		İ	length, head, and tail of the queue	

Exceptions:

The following table summarizes the exceptions required by this part and defined elsewhere in the package specification of Abstract_Data_Structures.Unbounded_- Priority_Queue:

1	Name	Ι	Descriptio	n]
	Queue Not Initialized		Indicates a	n	attempt	was	made	to	use	an	uninitialized	queue	

3.3.7.1.9.6.10.8.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why	Raised	*		
	Queue_Not_Initialized	Raised if uninitia	an attempt lized queue	is made	to manipulate a	in

3.3.7.1.9.6.10.9 DOT NEXT UNIT DESIGN

Given an input pointer P, this function returns the value of P.Next.
3.3.7.1.9.6.10.9.1 REQUIREMENTS ALLOCATION

None.

3.3.7.1.9.6.10.9.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.9.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	
	Ptr		Pointers		in		Pointer to the node whose "next" entry is to be returned	

3.3.7.1.9.6.10.9.4 LOCAL DATA

None.

3.3.7.1.9.6.10.9.5 PROCESS CONTROL

Not applicable.

.

3.3.7.1.9.6.10 9.6 PROCESSING

The following describes the processing performed by this part:

function Dot_Next (Ptr : in Pointers) return Pointers is
begin
 return Ptr.Next;
end Dot Next;

3.3.7.1.9.6.10.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top-level component:

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

1	Name	Туре	Range	Description	1
	Nodes	record	N/A	A single entity in the queue; contains data and a pointer to the next node	
Ĺ	Pointers	access	N/A	Points to a node in the queue	İ
	Queues	record 	N/A	Record containing the value of the current length, head, and tail of the queue	

```
3.3.7.1.9.6.10.9.8 LIMITATIONS
```

None.

3.3.7.1.9.6.10.10 SET NEXT UNIT DESIGN

Given an two input pointers, P and Q, this procedure sets P.Next equal to Q.

3.3.7.1.9.6.10.10.1 REQUIREMENTS ALLOCATION

None.

3.3.7.1.9.6.10.10.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.6.10.10.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name		Туре		Mode		Description	Ī
	Ptr		Pointers		in		Pointer to the node whose "next" entry is to be modified	
İ	Ptr_dot_Next	j	Pointers	İ	in	İ	Value to which Ptr.Next is to be set	1

3.3.7.1.9.6.10.10.4 LOCAL DATA

None.

3.3.7.1.9.6.10.10.5 PROCESS CONTROL

Not applicable.

```
3.3.7.1.9.6.10.10.6 PROCESSING
```

The following describes the processing performed by this part:

3.3.7.1.9.6.10.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top-level component:

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

.	Name		Туре		Range		Description	Ī
	Nodes		record		N/A		A single entity in the queue; contains data and a pointer to the next node	
	Pointers Queues		access record		N/A N/A		Points to a node in the queue Record containing the value of the current length, head, and tail of the queue	

3.3.7.1.9.6.10.10.8 LIMITATIONS

None.

3.3.7.1.9.7 BOUNDED STACK PACKAGE DESIGN (CATALOG #P335-0)

This generic package defines the data type and contains the operations required to perform last-in-first-out stacking operations on incoming data. The top of the stack always points to the last element added to the stack and the next element to be removed. When top equals 0, the stack is empty. When top equals Stack Size, the stack is full.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.9.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP require R166.



3.3.7.1.9.7.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.7.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of the Abstract_Data_Structures package:

Data types:

The following table summarizes the generic formal types required by this part:

	Name		Туре		Des	cription								
	Elements		private		User	defined	type	of	data	contained	in	the	stack	

Data objects:

The following table summarizes the generic formal objects required by this part:

	Name	1	Туре		Value		Description	Ī
	Initial Stack_Size		POSITIVE		N/A		Maximum number of elements which can be in the stack at any given time	Ī

3.3.7.1.9.7.4 LOCAL DATA

None.

3.3.7.1.9.7.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.7.6 PROCESSING

The following describes the processing performed by this part:

separate (Abstract_Data_structures)
package body Bounded_Stack is

end Bounded_Stack;

Page 1836

Q.

3.3.7.1.9.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following data types were previously defined in this part's package specification:

Name	Туре	Range	Description
Stack_ Length_ Range	POSITIVE subtype	1 Stack_Size	Used to dimension the list of elements
Stacks	limited private	N/A	List of data along with relevant information
Stack_ Statuses 	discrete type	Empty, Available, Full	Used to indicate the status of the stack

The following data types were previously defined in the private part of this part's package specification:

	Name	Type	Rang	Description	
	Stack_ Dimensions Range Stacks	Stack_ Dimensions subtype record	1 ' LAS' N/A	Used to dimension the list of elements List of data along with relevant information	

Data objects:

The following data objects were previously defined in this part's package specification:

Ι	Name	Туре	Value	Description	
	Stack_ Size	POSITIVE 	Initial Stack_Size	Number of elements in the stack	

Exceptions:

The following exceptions were previously defined in this part's package specification:

Page 1837

	Name	Description	
	Stack_Empty Stack_Full 	Error condition raised if an attempt is made to look at or retrieve elements from an empty stack Error condition raised if an attempt is made to add elements to a full stack	

3.3.7.1.9.7.8 LIMITATIONS

None.

3.3.7.1.9.7.9 LLCSC DESIGN

None.

3.3.7.1.9.7.10 UNIT DESIGN

3.3.7.1.9.7.10.1 CLEAR_STACK UNIT DESIGN

This procedure clears a stack by setting the top to 0.

3.3.7.1.9.7.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R166.

3.3.7.1.9.7.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.7.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name		Туре		Mode		Descrip	tion			
	Stack		Stacks		in out		Bounded	stack	being	manipulated	

3.3.7.1.9.7.10.1.4 LOCAL DATA

None.

Page 1838

3.3.7.1.9.7.10.1.5 PROCESS CONTROL Not applicable.

3.3.7.1.9.7.10.1.6 PROCESSING

The following describes the processing performed by this part:

procedure Clear Stack (Stack : out Stacks) is

begin

```
Stack.Top := 0;
```

end Clear Stack ;

3.3.7.1.9.7.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of the Abstract_Data_Structures.Bounded_Stack package:

Name		Туре		Range	Description	
Stack_ Length_ Range		POSITIVE subtype		1 Stack_Size	Used to dimension the list of elements	

The following data types were previously defined in the private part of this part's package specification:

Name	Туре	Range	Description	
Stacks	record	N/A 	List of data along with relevant information	

3.3.7.1.9.7.10.1.8 LIMITATIONS

None.



3.3.7.1.9.7.10.2 ADD_ELEMENT UNIT DESIGN This procedure adds an element to the top of the stack. A Stack_Full exception is raised if this routine is called with a full stack.

3.3.7.1.9.7.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R166.

3.3.7.1.9.7.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.7.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре		Mode		Description	1
	New_Element Stack	Elements Stacks		in in out		Element to be added to the stack Bounded stack being manipulated	

3.3.7.1.9.7.10.2.4 LOCAL DATA

None.

3.3.7.1.9.7.10.2.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.7.10.2.6 PROCESSING

The following describes the processing performed by this part:

procedure Add_Element (New_Element : in Elements; Stack : in out Stacks) is

-- -- declaration section

List : Lists renames Stack.List; Top : Stack_Length_Range renames Stack.Top;

-- -- begin procedure Add Element

į

begin

```
-- --make sure the stack is not already full
if Top = Stack_Size then
raise Stack_Full;
end if;
-- --add element to the stack
Top := Top + 1;
List(Top) := New_Element;
end Add Element ;
```

3.3.7.1.9.7.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

Ő

The following table summarizes the types required by this part and defined as generic formal parameters to the Abstract_Data_Structures.Bounded_Stack package:

	Name		Туре	I	Description								
1	Elements		private		User defined	type	of	data	contained	in	the	stack	

The following table summarizes the types required by this part and defined in the package specification of the Abstract_Data_Structures.Bounded_Stack package:

Ι	Name	I	Туре	1	Range		Description	
	Stack_ Length_ Range		POSITIVE subtype		1 Stack_Size		Used to dimension the list of elements	

The following data types were previously defined in the private part of this part's package specification:

Name	Type	Range	Description	
Lists Stacks 	array record	N/A N/A	List of elements List of data along with relevant information	

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Abstract Data Structures. Bounded Stack package:

	Name		Туре		Value		Description	
	Stack_ Size		POSITIVE		Initial Stack_Size		Number of elements in the stack	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_Stack package:

	Name	Description	
	Stack_Full	Error condition raised if an attempt is made to add elements to a full stack	

3.3.7.1.9.7.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name		When/W	lhy	Rai	ised									,	1
	Stack_Full		Raised	if	an	attempt	is	made	to	add	elements	to	a	full	stack	Ī

3.3.7.1.9.7.10.3 RETRIEVE ELEMENT UNIT DESIGN

This procedure retrieves the top element from the stack and returns it to the calling routine.

A Stack_Empty exception is raised if this routine is called with an empty stack.

3.3.7.1.9.7.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R166.

3.3.7.1.9.7.10.3.2 LOCAL ENTITIES DESIGN

None.

```
3.3.7.1.9.7.10.3.3 INPUT/OUTPUT
```

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name		Туре	1	Mode		Description	
	Stack Old_Element		Stacks Elements		in out out	 	Bounded stack being manipulated Element retrieved from the stack	

3.3.7.1.9.7.10.3.4 LOCAL DATA

None.

3.3.7.1.9.7.10.3.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.7.10.3.6 PROCESSING

The following describes the processing performed by this part: .

procedure Retrieve_Element (Stack : in out Stacks; Old Element : out Elements) is

-- -- declaration section

List : Lists renames Stack.List; Top : Stack_Length_Range renames Stack.Top;

-- --begin procedure Retrieve_Element

begin

•

,

```
-- --make sure there is something in the stack to retrieve
if Top = 0 then
raise Stack_Empty;
end if;
-- --retrieve and remove the top element from the stack
Old_Element := List(Top);
Top := Top - 1;
```

end Retrieve_Element ;

Page 1844

3.3.7.1.9.7.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Abstract_Data_Structures.Bounded_Stack package:

	Name		Туре		Description								
	Elements		private		User defined	type	of	data	contained	in	the	stack	

The following table summarizes the types required by this part and defined in the package specification of the Abstract_Data_Structures.Bounded_Stack package:

Name		Туре		Range		Description	1
Stack_ Length_ Range	1	POSITIVE subtype		1 Stack_Size		Used to dimension the list of elements	

The following data types were previously defined in the private part of this part's package specification:

	Name	Туре	Range	Description	
	Lists Stacks	array record	N/A N/A	List of elements List of data along with relevant information	

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_Stack package:

	Name		Туре	Value	I	Description	
	Stack_ Size		POSITIVE	Initial Stack_Size		Number of elements in the stack	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_Stack package:

1	Name	1	Description	
 	Stack_Empty		Error condition raised if an attempt is made to look at or retrieve elements from an empty stack	

3.3.7.1.9.7.10.3.8 LIMITATIONS

The following table describes the exceptions raised by this part:

1	Name		When/Why Raised	Ī
	Stack_Empty		Raised if an attempt is made to look at or retrieve elements from an empty stack	

3.3.7.1.9.7.10.4 PEEK UNIT DESIGN

This function returns the data in the top element of the stack, but does not remove the element from the stack.

A Stack_Empty exception is raised if this routine is called with an empty stack.

3.3.7.1.9.7.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R166.

3.3.7.1.9.7.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.7.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре		Mode		Description	•
S	tack	!	Stacks		in out		Bounded stack being manipulated	



3.3.7.1.9.7.10.4.4 LOCAL DATA

None.

3.3.7.1.9.7.10.4.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.7.10.4.6 PROCESSING

The following describes the processing performed by this part:

function Peek (Stack : in Stacks) return Elements is

List : Lists renames Stack.List; Top : Stack_Length_Range renames Stack.Top;

-- --begin function Peek

-- -----

begin

```
-- --make sure there is something in the stack
if Top = 0 then
    raise Stack_Empty;
end if;
```

-- --return value in top element of the stack return List(Top);

end Peek ;

3.3.7.1.9.7.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Abstract_Data_Structures.Bounded_Stack package:

.

.

	Name	;	Туре	1	Desc	cription								
	Elements		private	I	User	defined	type	of	data	contained	in	the	stack	

The following table summarizes the types required by this part and defined in the package specification of the Abstract_Data_Structures.Bounded_Stack package:

Name	Туре	Range	Description	
Stack_ Length_ Range	POSITIVE subtype 	1 Stack_Size	Used to dimension the list of elements 	

The following data types were previously defined in the private part of this part's package specification:

Name	Туре	Range	Description	
Lists Stacks 	array record	N/A N/A	List of elements List of data along with relevant information	

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_Stack package:

	Name		Туре		Value	Ī	Description	
	Stack		POSITIVE		Initial Stack_Size		Number of elements in the stack	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_Stack package:

	Name	Description	Ī
	Stack_Empty 	Error condition raised if an attempt is made to look at or retrieve elements from an empty stack	

3.3.7.1.9.7.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

 Name	When/Why Raised	
Stack_Empty	Raised if an attempt is made to look at or retrieve elements from an empty stack	

3.3.7.1.9.7.10.5 STACK STATUS UNIT DESIGN

This function returns the status of the stack based on the following algorithm:

if no elements are in the stack then stack status is empty elsif the maximum number of elements are in the stack then stack status is full else stack status is available end if

3.3.7.1.9.7.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R166.

3.3.7.1.9.7.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.7.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name		Туре		Mode			Description			
	Stack		Stacks	I	in ou	:]	Bounded stack	being	manipulated	

3.3.7.1.9.7.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name		Туре		Value	1	Description	
	Status	1	Stack_Statuses		N/A		Status of the stack	

Page 1848

- A

ł

```
3.3.7.1.9.7.10.5.5 PROCESS CONTROL Not applicable.
```

3.3.7.1.9.7.10.5.6 PROCESSING

The following describes the processing performed by this part:

function Stack Status (Stack : in Stacks) return Stack Statuses is

-- --declaration section -- --declaration section Top : Stack_Length_Range renames Stack.Top; Status : Stack_Statuses; -- --begin function Stack Status

begin

```
if Top = 0 then
   Status := Empty;
elsif Top = Stack Size then
   Status := Full;
else
   Status := Available;
end if;
```

return Status;

end Stack Status ;

3.3.7.1.9.7.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

.

•

The following table summarizes the types required by this part and defined in the package specification of the Abstract_Data_Structures.Bounded_Stack package:

143

Name	Туре	Range	Description	
Stack_ Length_ Range	POSITIVE subtype	1 Stack_Size	Used to dimension the list of elements	
Stack_ Statuses 	discrete type	Empty, Available, Full	Used to indicate the status of the stack	

The following data types were previously defined in the private part of this part's package specification:

Name	Туре	Range	Description	
Stacks	record	N/A	List of data along with relevant information	

Data objects:

The following table summarizes the objects required by this part and defined in the package specification of Abstract_Data_Structures. Bounded_Stack package:

	Name	Type	Value	Description	
	Stack_ Size	POSITIVE	Initial Stack_Size	Number of elements in the stack	

3.3.7.1.9.7.10.5.8 LIMITATIONS

None.

3.3.7.1.9.7.10.6 STACK LENGTH UNIT DESIGN

This function returns the length of the stack.

3.3.7.1.9.7.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R166.

3.3.7.1.9.7.10.6.2 LOCAL ENTITIES DESIGN

None.

Page 1850

3.3.7.1.9.7.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Type		Mode		Description	
	Stack	Stacks	1	in out		Bounded stack being manipulated	

3.3.7.1.9.7.10.6.4 LOCAL DATA

None.

3.3.7.1.9.7.10.6.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.7.10.6.6 PROCESSING

The following describes the processing performed by this part:

function Stack Length (Stack : in Stacks) return Stack Length Range is

begin

return Stack.Top;

end Stack_Length ;

3.3.7.1.9.7.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of the Abstract Data Structures.Bounded Stack package:

1	Name		Туре		Range		Description	1
	Stack Length Range		POSITIVE subtype		1 Stack_Size		Used to dimension the list of elements	





The following data types were previously defined in the private part of this part's package specification:

	Name	Туре	Range	Description	
	Stacks	record 	N/A 	List of data along with relevant information	

3.3.7.1.9.7.10.6.8 LIMITATIONS

None.

3.3.7.1.9.8 UNBOUNDED STACK PACKAGE DESIGN (CATALOG #P336-0)

This generic package performs last-in-first-out stacking operations on incoming data. The head of the stack always points to the last element added to the stack and the next element to be removed. The tail always points to a dummy node located below the oldest element on the stack. If head and tail point to the same node, the stack is empty.

An available space list is maintained local to this part. When this part is elaborated the available space list will have a dummy node plus Initial -Available Space_Size nodes. When nodes are added to the stack, the Add_Element routine will try to get a node from the available space list before attempting to allocate more memory. When the Retrieve Element routine is called, the unused node will be returned to the available space list for later use. The memory committed to the available space may be deallocated by calling the Free Memory procedure.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.7.1.9.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.2 LOCAL ENTITIES DESIGN

Data structures:

This part maintains an available space list local to the package body.

Subprograms:

The following subprograms are contained local to this body:

Page 1852

Name	Type	Description	
Free Node Dot_Next	procedure function	Instantiation of UNCHECKED_DEALLOCATION Given a pointer P, this function returns the value of P.Next	
Set_Next 	procedure	Given two points P & Q, this procedure sets P.Next = Q	

The following subprograms are contained in this part as a result of renaming operations on identically named routines contained in the locally instantiated Available_Space_Operations package.

	Name	Туре	Description	1
	New_Node Save_Node Save_List	function procedure procedure	Returns a node to the calling routine; will get a node from the available space list if possible, otherwise will allocate a new node Handles placing a node in the available space list Handles placing a list of nodes in the available space list	

This package body contains code to initialize the Available Space List. This code is executed when the package is elaborated. At a minimum, this code calls the Initialize procedure to initialize the Available Space List so it contains a dummy node. If the generic formal object Initial Available Space Size is greater than or equal to 1, this routine then places the requested number of nodes (in addition to the dummy node) in the available space list.

3.3.7.1.9.8.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified in the package specification of the Abstract Data Structures package:

Data types:

The following table summarizes the generic formal types required by this part:

	Name		Туре	1	Desc	cription								 I
	Elements		private	1	User	defined	type	of	data	contained	in	the	stack	

Data objects:

The following table summarizes the generic formal objects required by this part:

6

C.

Name	Туре	Description
Initial_Available_	NATURAL	Number of nodes to be initially placed in
	!	I the available space list
	Δ Ψ.Δ	
	nia	
one.		
.3.7.1.9.8.5 PROCESS	CONTROL	
ot applicable.		
ne following describes	s the proce	essing performed by this part:
ith UNCHECKED_DEALLOCA	ATION; a Structure))
ackage body Unbounded	Stack is	53 /
declaration section	ac 	
abij stati doji	A A A A A A A A A A A A A A A A A A A	Weiken station of the August his door
Initial_Head : Point	ccessed UNL ters := new	Nodes;
Available Space : Si	tacks := (C	Current Length => 0,
_ •	T	<pre>cop => Initial Head,</pre>
		- mitiai_head);
Available_Length : Available_Top : H	Pointers re	names Available_Space.Current_Length; names Available_Space.Top;
Available_Bottom : H	Pointers re	names Available_Space.Bottom;
procedure Free is ne	W UNCHECKE	D_DEALLOCATION
	Name	=> Pointers);
procedure Free_Node renames Fr	(Which_Nod	e : in out Pointers)
function Dot_Next (H	?tr : in Po	inters) return Pointers;
procedure Set Next (Ptr	: in Pointers;
-	Ptr_dot_Ne	xt : in Pointers);
package Available_Sp	ace_Operat	ions is new
Available_Sp (Nodes	ace_List_0	<pre>perations</pre>
Pointers		=> Pointers.

```
Available Length
                                      => Available Length,
               Available Head
                                       => Available Top,
               Available Tail
                                       => Available Bottom);
   function New Node return Pointers
            renames Available Space Operations.New Node;
   procedure Save Node (Saved Node : in Pointers)
            renames Available Space Operations.Save Node;
   procedure Save List (Saved Head : in Pointers;
                        Saved Tail : in Pointers;
                        Node Count : in POSITIVE)
            renames Available Space Operations.Save List;
--begin package Unbounded Stack
--(see header for package body for details)
begin
-- -- set up available space list if one is desired
   if Initial_Available_Space_Size > 0 then
      Add_Nodes_to_Available_Space_List:
         for I In I..Initial Available Space Size loop
            Available_Bottom.Next := NEW Nodes;
            Available Bottom := Available Bottom.Next;
         end loop Add Nodes to Available Space List;
      Available_Length := Initial_Available_Space_Size;
   end if;
end Unbounded Stack;
3.3.7.1.9.8.7 UTILIZATION OF OTHER ELEMENTS
Subprograms and task entries:
The following table describes the subroutines required by this part:
```

	Name	Туре	Source	Description	
	UNCHECKED DEALLOCATION	generic function	N/A	Used to deallocate memory	

Exceptions:

38

The following table describes the exceptions required by this part and defined in the Ada predefined package STANDARD:

	Name		Description	Ī
	STORAGE_ERROR		Raised when an attempt is made to dynamically allocate more memory than is available	

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table describes the packages required by this part and specified in the package body of the Abstract Data Structures package:

	Name		Туре		Description	
	Available_Space_ List_Operations		generic package		Contains the routines required to retrieve a node from and place a node in the available space list	

Data types:

The following data types were previously defined in this part's package specification:

1	Name	Ι	Туре		Range		Description	
 	Stack Statuses		discrete type		Empty, Available Uninitialized		Indicates the current status of the stack	

The following table describes the data types defined in the private part of the Abstract Data Structures.Unbounded Stack package:

	Name	Туре	Range		Description	
	Nodes Pointers	record access	N/A N/A		Contains a single element and a pointer to another node Points to & node	
	Stacks	record 	N/A 		List of data along with relevant information	

Exceptions:

The following exceptions were previously defined in this part's package specification:

Page	1857

	Name	Description	
	Stack_Empty Stack_Not_ Initialized	Error condition raised if an attempt is made to look at or retrieve elements from an empty stack Raised if an attempt is made to use an uninitialized stack	

3.3.7.1.9.8.8 LIMITATIONS

The following table describes the exceptions raised by this part:

1	Name	When/Why Raised	Ī
	STANDARD.STORAGE_ERROR	Raised during elaboration of this package if an attempt is made to allocate memory when no more is available	

3.3.7.1.9.8.9 LLCSC DESIGN

None.

3.3.7.1.9.8.10 UNIT DESIGN

3.3.7.1.9.8.10.1 INITIALIZE UNIT DESIGN

This procedure initializes a stack by placing a dummy node in the stack, pointing the top and bottom to the dummy node, and setting the length to 0. If this routine is called with a stack containing elements, then the stack is cleared of all but the dummy node.

3.3.7.1.9.8.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ċ

	Type Mode Description
Stack	Stacks in out Stack being manipulated
3.3.7.1.9.8.10.	1.4 LOCAL DATA
None.	
3 3 7 1 9 8 10	1 5 PROCESS CONTROL
Net epoliceble	
Not applicable.	
3.3.7.1.9.8.10.	1.6 PROCESSING
The following de	escribes the processing performed by this part:
procedure In:	itialize (Stack : in out Stacks) is
declara	tion section
Current L	ength : INTEGER renames Stack Current Length:
Top Bottom	: Pointers renames Stack.Top;
Bottom	; rolliters relianes Stack. Bottom;
begin proce	edure Initialize
begin	
if Current	$L_{\rm Length} = -1$ then
handl	ie an uninitialized stack :- New Node:
Bottom	:= Top;
	Length := 0;
eisii Curi	rent_Lengtn > 0 then
hand Clear_S	le a stack that has elements in it Stack (Stack => Stack);
else	
curre	ent length = 0, so do nothing
NULL;	
end if;	
end Initializ	2e ;

3.3.7.1.9.8.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines required by this part and defined in the package specification of Abstract_Data_Structures.Unbounded_- Stack:

i	Name	Туре	Description	-
	Clear_Stack	procedure	Clears a stack by returning all of its nodes to the available space list	

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Abstract_Data_-Structures.Unbounded Stack:

Name		Туре		Description	
New_Node		function		Returns a node to the calling routine; will get a node from the available space list if possible, otherwise will allocate a new node	

Data types:

The following table describes the data types defined in the private part of the Abstract Data Structures.Unbounded Stack package:

	Name	Туре	Range		Description	
	Nodes	record	N/A		Contains a single element and a pointer to another node	
	Pointers Stacks	access record	N/A N/A		Points to a node List of data along with relevant information	

3.3.7.1.9.8.10.1.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why Raised	
 	STANDARD.STORAGE_ERROR	Raised if an attempt is made to allocate mor memory than is available	e

3.3.7.1.9.8.10.2 CLEAR STACK UNIT DESIGN

This procedure removes nodes from a stack, leaving only the dummy node. The nodes removed are placed in the available space list.

A Stack Not Initialized exception is raised if this routine is called with an uninitialized stack.

3.3.7.1.9.8.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	Тур	:	Mode		Description	
	Stack	Stac	.s	in out		Stack being manipulated	1

3.3.7.1.9.8.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name		Туре	1	Value		Description	-
	This_Node		Pointers		N/A		Points to the node to be placed in the available space list	

1

٠

.

3.3.7.1.9.8.10.2.5 PROCESS CONTROL Not applicable. 3.3.7.1.9.8.10.2.6 PROCESSING The following describes the processing performed by this part: procedure Clear_Stack (Stack : in out Stacks) is --declaration section ----------Current Length : INTEGER renames Stack.Current Length; : Pointers renames Stack.Top: Top Bottom : Pointers renames Stack.Bottom: This Node : Pointers; -- -- begin procedure Clear Stack ______ begin --make sure stack has been initialized --if Current Length = -1 then raise Stack Not Initialized; --make sure there is something in the stack elsif Current Length /= 0 then --placed nodes in the available space list --Save_List (Saved_Head => Top.Next, Saved Tail => Bottom, Node Count => Current Length); --reinitialize stack variables Top.Next := NULL Bottom := Top; := NULL: Current Length := 0; end if; end Clear Stack ; 3.3.7.1.9.8.10.2.7 UTILIZATION OF OTHER ELEMENTS UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Abstract_Data_-Structures.Unbounded Stack:

	Name		Туре		Description	1
	Save_List		procedure		Handles placing a list of nodes in the available space list	

Data types:

The following table describes the data types defined in the private part of the Abstract Data Structures.Unbounded Stack package:

Ī	Name	Туре	Range	Description	
	Nodes Pointers	record access	N/A N/A	Contains a single element and a pointer to another node Points to a node	
	Stacks	record	N/A	List of data along with relevant information	İ

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract Data Structures. Unbounded Stack:

	Name		Descri	pti	on							
	Stack Not Initialized		Raised	if	an	attempt	is	made	to	use an	uninitialized	stack

3.3.7.1.9.8.10.2.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	I	When/Why Raised	-
	Stack_Not_Initialized		Raised if an attempt is made to manipulate an uninitialized stack	

3.3.7.1.9.8.10.3 FREE MEMORY UNIT DESIGN

This procedure deallocates the memory occupied by the nodes in the available space list. Only a dummy node will be left in the list.

3.3.7.1.9.8.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.3.3 INPUT/OUTPUT

None.

3.3.7.1.9.8.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

```
Ć
```

	Name		Туре		Value		Description	I
	This_Node		Pointers		N/A		Points to the node to be deallocated	Ī

3.3.7.1.9.8.10.3.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.8.10.3.6 PROCESSING

The following describes the processing performed by this part:

procedure Free Memory is

This Node : Pointers;

-- --begin procedure Free_Memory

begin

```
Deallocate Nodes in Available Space List:
       while Available Top /= Available Bottom loop
          This Node := Available Top;
          Available Top := Available Top.Next;
          Free Node (Which Node => This Node);
       end loop Deallccate Nodes in Available Space List;
     Available Length := 0;
     Available Top.Next := NULL;
  end Free Memory ;
3.3.7.1.9.8.10.3.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:
The following tables describe the elements used by this part but defined
elsewhere in the parent top level component:
Subprograms and task entries:
The following table summarizes the subroutines and task entries required by
this part and defined in the package body of Abstract Data -
Structures.Unbounded_Stack:
                         Name Type Description
```

| Free_Node | procedure | Instantiation of UNCHECKED_DEALLOCATION |

Data types:

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Unbounded_Stack package:

	Name	Туре	Range		Description
	Nodes Pointers Stacks	record access record	N/A N/A N/A		Contains a single element and a pointer to another node Points to a node List of data along with relevant information

3.3.7.1.9.8.10.3.8 LIMITATIONS

None.

6

3.3.7.1.9.8.10.4 ADD ELEMENT UNIT DESIGN

This procedure adds an element to the top of the stack.

A Stack Not Initialized exception is raised if this routine is called with an uninitialized stack.

3.3.7.1.9.8.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name	Туре	Mode	Description	
	New_Element Stack	Elements Stacks	in in out	Element to be added to the stack Stack being manipulated	

3.3.7.1.9.8.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name		Туре		Value		Description	Ī
	Ptr		Pointers		N/A		Points to the new node to be placed in the stack	

3.3.7.1.9.8.10.4.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.8.10.4.6 PROCESSING

The following describes the processing performed by this part:

procedure Add_Element (New_Element : in Elements; Stack : in out Stacks) is

```
_____
    --declaration section
--
     _____
___
     Current Length : INTEGER renames Stack.Current Length;
     Top : Pointers renames Stack.Top;
     Ptr
                : Pointers:
__ ____
-- -- begin procedure Add Element
______
  begin
     if Current Length = -1 then
       raise Stack Not Initialized;
     end if;
     --get a node and initialize it
_ --
     Ptr := New Node;
     Ptr.Data := New Element;
     --place the node on the stack
-----
     Ptr.Next := Top;
Top := Ptr;
     Current Length := Current_Length + 1;
  end Add Element ;
3.3.7.1.9.8.10.4.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:
The following tables describe the elements used by this part but defined elsewhere in the parent top level component:
Subprograms and task entries:
The following table summarizes the subroutines and task entries required by
this part and defined in the package body of Abstract_Data_-
Structures.Unbounded Stack:
| Name | Type | Description
_____
 New Node | function | Returns a node to the calling routine; will get a
```

node from the available space list if possible, otherwise will allocate a new node

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Abstract_Data_Structures. Unbounded_Stack package:

	Name	Туре	De	scription							1
	Elements	private	Use	r defined	type of	data	contained	in	the	stack	

The following table describes the data types defined in the private part of the Abstract Data Structures.Unbounded Stack package:

	Name	Туре	Range		Description	-
	Nodes Pointers Stacks	record access record	N/A N/A N/A		Contains a single element and a pointer to another node Points to a node List of data along with relevant information	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract Data Structures. Unbounded Stack:

	Name	1	Descri	ptio	n						Ī
	Stack Not Initialized	1	Raised	if a	n attempt	is made	toı	use an	uninitialized	stack	

3.3.7.1.9.8.10.4.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why Raised	
	STANDARD.STORAGE_ERROR	Raised if an attempt is made to allocate more memory than is available	
	Stack_Not_Initialized	Raised if an attempt is made to manipulate an uninitialized stack	

3.3.7.1.9.8.10.5 RETRIEVE ELEMENT UNIT DESIGN

This procedure retrieves the top element of the stack and returns the data in it to the calling routine. The node is then placed in the available space list.

A Stack_Empty exception is raised if this routine is called with an empty stack.

A Stack_Not_Initialized exception is raised if this routine is called with an uninitialized stack.

3.3.7.1.9.8.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре	I	Mode		Description	
	Stack Old_Element		Stacks Elements		in out out		Stack being manipulated Elements retrieved from the stack	

3.3.7.1.9.8.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name	Туре	Value	Description	
	This_Node	Pointers	N/A 	Node to be returned to the available space list	

3.3.7.1.9.8.10.5.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.8.10.5.6 PROCESSING

The following describes the processing performed by this part:

procedure Retrieve_Element (Stack : in out Stacks; Old Element : out Elements) is
26

esta.

.

•

declaration section
Current_Length : INTEGER renames Stack.Current_Length; Top : Pointers renames Stack.Top;
This_Node : Pointers;
begin procedure Retrieve_Element
begin
<pre>make sure there is something to retrieve if Current Length = -1 then raise Stack Not Initialized; elsif Current Length = 0 then raise Stack_Empty; end if;</pre>
retrieve data in the top node Old_Element := Top.Data;
dispose of top node and adjust the stack This_Node := Top; Top := Top.Next; Save_Node (Saved_Node => This_Node); Current_Length := Current_Length - 1;
end Retrieve_Element ;
3.3.7.1.9.8.10.5.7 UTILIZATION OF OTHER ELEMENTS
UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:
The following tables describe the elements used by this part but defined elsewhere in the parent top level component:
Subprograms and task entries:
The following table summarizes the subroutines and task entries required

The following table summarizes the subroutines and task entries required by this part and defined in the package body of Abstract_Data_-Structures.Unbounded_Stack:

Ī	Name		Туре		Descri	ption								
	Save_Node		procedure		Handles	placing	a	node	in	the	available	space	list	

Data types:



The following table summarizes the types required by this part and defined as generic formal parameters to the Abstract_Data_Structures. Unbounded_Stack package:

	Name		Туре		Dese	cription								
	Elements		private	Ι	User	defined	type	of	data	contained	in	the	stack	

The following table describes the data types defined in the private part of the Abstract_Data_Structures.Unbounded_Stack package:

1	Name	Туре	Range		Description	
	Nodes	record	N/A		Contains a single element and a pointer to another node	
	Pointers Stacks	access record	N/A N/A		Points to a node List of data along with relevant information	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract Data Structures. Unbounded Stack:

I	Name		Description	
	Stack_Empty Stack_Not_ InitTalized		Error condition raised if an attempt is made to look at or retrieve elements from an empty stack Raised if an attempt is made to use an uninitialized stack	

3.3.7.1.9.8.10.5.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why Raised	I
	Stack_Not_Initialized	Raised if an attempt is made to manipulate an uninitialized stack	
 	Stack_Empty	Raised if an attempt is made to retrieve or look at elements in an empty stack	

3.3.7.1.9.8.10.6 PEEK UNIT DESIGN

This function returns the data contained in the top element of the stack, but does not remove the element from the stack.

A Stack_Empty exception is raised if this routine is called with an empty stack.

A Stack Not Initialized exception is raised if this routine is called with an uninitialized stack.

3.3.7.1.9.8.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Ī	Name		Туре		Mode		Description	
	Stack	·I	Stacks		in out		Stack being manipulated	

3.3.7.1.9.8.10.6.4 LOCAL DATA

None.

3.3.7.1.9.8.10.6.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.8.10.6.6 PROCESSING

The following describes the processing performed by this part:

function Peek (Stack : in Stacks) return Elements is

-- --declaration section -- --declaration section -- Current_Length : INTEGER renames Stack.Current_Length; Top : Pointers renames Stack.Top; -- --begin function Peek





begin

 make sure there is something to peek at
if Current Length = -1 then
raise Stack Not Initialized;
elsif Current \overline{L} ength = 0 then
raise Stack Empty;
end if;

-- --returned desired element return Top.Data;

end Peek ;

3.3.7.1.9.8.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Abstract_Data_Structures. Unbounded_Stack package:

	Name		Туре		Dese	cription								
	Elements		private		User	defined	type	of	data	contained	in	the	stack	

The following table describes the data types defined in the private part of the Abstract Data Structures.Unbounded Stack package:

	Name	Туре	Range		Description	Ī
	Nodes	record	N/A		Contains a single element and a pointer to another node	
	Pointers Stacks	access record	N/A N/A 		List of data along with relevant information	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract_Data_Structures. Unbounded Stack:

 	Name	 I	Description
 	Stack_Empty Stack_Not_ Initialize	2 	Error condition raised if an attempt is made to look at or retrieve elements from an empty stack Raised if an attempt is made to use an uninitialized stack

3.3.7.1.9.8.10.6.8 LIMITATIONS

The following table describes the exceptions raised by this part:

Name	When/Why Raised	
Stack_Not_Initialized	Raised if an attempt is made to manipulate an uninitialized stack	
Stack_Empty	Raised if an attempt is made to retrieve or look at elements in an empty stack	İ

3.3.7.1.9.8.10.7 STACK STATUS UNIT DESIGN

This function returns the status of the stack according to the following algorithm:

if stack has never been initialized then stack status is uninitialized elsif stack has no elements in it then stack status in empty else stack status is available end if

3.3.7.1.9.8.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Туре	Mode Description	
Stack	Stacks	in out Stack being manipulated	

I

3.3.7.1.9.8.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

	Name		Туре		Value		Description	-
	Status		Stack_Statuses		N/A		Status of the stack	

3.3.7.1.9.8.10.7.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.8.10.7.6 PROCESSING

The following describes the processing performed by this part:

function Stack_Status (Stack : in Stacks) return Stack Statuses is

-- -- declaration section

Current_Length : INTEGER renames Stack.Current_Length;

Status : Stack_Statuses;

-- --begin function Stack_Status

begin

```
if Current_Length = -1 then
    Status := Uninitialized;
elsif Current_Length = 0 then
    Status := Empty;
else
    Status := Available;
end if;
return Status;
end Stack_Status ;
```

Page 1874

3.3.7.1.9.8.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined in the package specification of Abstract Data Structures. Unbounded Stack:

	Name	1	Туре		Range		Description	
	Stack_ Statuses		discrete type		Empty, Available Uninitialized		Indicates the current status of the stack	

The following table describes the data types defined in the private part of the Abstract_Data Structures.Unbounded Stack package:

	Name	Туре	Range	Description	
	Stacks	record 	N/A 	List of data along with relevant information	

3.3.7.1.9.8.10.7.8 LIMITATIONS

None.

3.3.7.1.9.8.10.8 STACK LENGTH UNIT DESIGN

This function returns the length of the stack.

A Stack Not Initialized exception is raised if this routine is called with an uninitialized stack.

3.3.7.1.9.8.10.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R167.

3.3.7.1.9.8.10.8.2 LOCAL ENTITIES DESIGN

None.

Page 1875

Page 1876

3.3.7.1.9.8.10.8.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	Туре		Mode		Description	-
	Stack	Stacks	:	in out	1	Stack being manipulated	

3.3.7.1.9.8.10.8.4 LOCAL DATA

None.

.

3.3.7.1.9.8.10.8.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.8.10.8.6 PROCESSING

The following describes the processing performed by this part:

function Stack Length (Stack : in Stacks) return NATURAL is

-- -- declaration section

Current Length : INTEGER renames Stack.Current Length;

-- --begin function Stack_Length

begin

```
-- --make sure stack has been initialized
if Current_Length = -1 then
    raise Stack_Not_Initialized;
end if;
return Current_Length;
```

end Stack_Length ;

3.3.7.1.9.8.10.8.7 UTILIZATION OF OTHER ELEMENTS UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table describes the data types defined in the private part of the Abstract Data Structures.Unbounded_Stack package:

Name	Туре	Range	Description	
Stacks	record	N/A	List of data along with relevant information	

Exceptions:

The following table summarizes the exceptions required by this part and defined in the package specification of Abstract Data Structures. Unbounded Stack:

	Name		Descri	ptio	 1								
	Stack Not Initialized		Raised	if a	n attempt	is	made	to	use	an	uninitialized	stack	

3.3.7.1.9.8.10.8.8 LIMITATIONS

The following table describes the exceptions raised by this part:

	Name	When/Why Raised	1
	Stack_Not_Initialized	Raised if an attempt is made to manipulate an uninitialized stack	

3.3.7.1.9.8.10.9 DOT NEXT UNIT DESIGN

Given an input pointer P, this function returns the value of P.Next.

3.3.7.1.9.8.10.9.1 REQUIREMENTS ALLOCATION

None.

3.3.7.1.9.8.10.9.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.9.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

	Name		Туре	1	Mode		Description	
	Ptr		Pointers		in		Pointer to the node whose "next" entry is to be returned	

3.3.7.1.9.8.10.9.4 LOCAL DATA

None.

3.3.7.1.9.8.10.9.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.8.10.9.6 PROCESSING

The following describes the processing performed by this part:

function Dot_Next (Ptr : in Pointers) return Pointers is
begin
 return Ptr.Next;
end Dot_Next;

3.3.7.1.9.8.10.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top-level component:

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

1	Name	Туре	Range	Description
	Nodes	record	N/A	A single entity in the stack; contains data and a pointer to the next node
	Pointers Stacks	access record	N/A N/A	Points to a node in the stack Record containing the value of the current length, head, and tail of the stack

Page 1878



```
3.3.7.1.9.8.10.9.8 LIMITATIONS None.
```

3.3.7.1.9.8.10.10 SET NEXT UNIT DESIGN

Given an two input pointers, P and Q, this procedure sets P.Next equal to Q.

3.3.7.1.9.8.10.10.1 REQUIREMENTS ALLOCATION

None.

3.3.7.1.9.8.10.10.2 LOCAL ENTITIES DESIGN

None.

3.3.7.1.9.8.10.10.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

```
      Name
      Type
      Mode
      Description

      Ptr
      Pointers
      in
      Pointer to the node whose "next" entry

      I
      is to be modified
      is to be modified

      Ptr_dot_Next
      Pointers
      in
      Value to which Ptr.Next is to be set
```

3.3.7.1.9.8.10.10.4 LOCAL DATA

None.

3.3.7.1.9.8.10.10.5 PROCESS CONTROL

Not applicable.

3.3.7.1.9.8.10.10.6 PROCESSING

The following describes the processing performed by this part:

3.3.7.1.9.8.10.10.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP-LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top-level component:

Data types:

The following table summarizes the types required by this part and defined in the private portion of the part's package specification:

1	Name	Туре	Range	1	Description	1
	Nodes	record	N/A		A single entity in the stack; contains data and a pointer to the next node	
	Pointers Stacks	access record	N/A N/A		Points to a node in the stack Record containing the value of the current length, head, and tail of the stack	

3.3.7.1.9.8.10.10.8 LIMITATIONS

None.

3.3.7.1.10 UNIT DESIGN

None.

```
Page 1881
```

```
package body Abstract Data Structures is
pragma PAGE;
   generic
      type Nodes
                            is limited private;
      type Pointers
                            is access Nodes;
      Available Length
                            : in out INTEGER;
      Available Head
                             : in out Pointers:
      Available Tail
                             : in out Pointers;
      with function Dot Next (Ptr : in Pointers) return Pointers is <>;
      with procedure Set Next (Ptr
                                          : in Pointers;
                              Ptr Dot Next : in Pointers) is <>;
   package Available_Space_List_Operations is
      function New Node return Pointers;
      procedure Save Node (Saved Node : in Pointers);
      procedure Save List (Saved Head : in Pointers;
                          Saved Tail : in Pointers;
                          Node Count : in POSITIVE);
   end Available_Space_List_Operations;
pragma PAGE;
-- -- separate package bodies
__ _____
   package body Bounded Fifo Buffer is separate;
   package body Unbounded Fifo Buffer is separate;
   package body Nonblocking Circular Buffer is separate;
   package body Unbounded_Priority_Queue is separate;
   package body Bounded Stack is separate;
   package body Unbounded Stack is separate;
   package body Available Space List Operations is separate;
end Abstract Data Structures;
```

separate (Abstract_Data_Structures)
package body Available_Space_List_Operations is

pragma PAGE;

function New Node return Pointers is

-- -- declaration section

Ptr : Pointers; New_Available_Head : Pointers;

__ ____

-- -- begin function New Node

begin

if Available_Length > 0 then

-- -- get the node from the available space list and mark the node
-- -- that will now be the head of the available space list
Ptr := Available_Head;
New_Available_Head := Dot_Next(Available_Head);
-- -- initialize node being returned
Set_Next (Ptr => Ptr,
Ptr_Dot_Next => null);

```
-- -- adjust the available space list
Available Head := New Available Head;
Available_Length := Available_Length - 1;
```

else

--allocate space to get the node
Ptr := new Nodes;

end if;

return Ptr;

end New Node;

.

pragma PAGE;
 procedure Save Node(Saved_Node : in Pointers) is

begin

Set_Next (Ptr => Available_Tail, Ptr_Dot_Next => Saved_Node); Available_Tail := Saved_Node; Set_Next (Ptr => Available_Tail, Ptr_Dot_Next => null); Page 1882

```
Available_Length := Available_Length + 1;
   end Save Node;
pragma PAGE;
   procedure Save List (Saved Head : in Pointers;
                        Saved Tail : in Pointers;
                        Node Count : in POSITIVE) is
  begin
      Set Next (Ptr
                               => Available Tail,
                Ptr_Dot_Next => Saved_Head);
      Available_Tail := Saved_Tail;
      Set Next (Ptr
                               => Available Tail,
                Ptr Dot Next => Saved Head);
      Available Length := Available Length + Node Count;
   end Save List;
end Available_Space_List_Operations;
```

Page 1883

CAMP Software Detailed Design Document separate (Abstract Data Structures) package body Bounded Fifo Buffer is pragma PAGE; procedure Clear Buffer (Buffer : out Buffers) is ___ -- declaration section _____ ___ Buffer Length : Buffer Range renames Buffer.Buffer Length; Head : Buffer Range renames Buffer.Head; : Buffer Range renames Buffer.Tail; Tail __ ____ -- -- begin procedure Clear Buffer begin Buffer_Length := 0; Head := 0; := 1; Tail end Clear Buffer ; pragma PAGE; procedure Add_Element (New_Element : in Elements; Buffer : in out Buffers) is ___ -- declaration section -----LIST : Lists renames Buffer.LIST; Buffer Length : Buffer Range renames Buffer.Buffer Length; Head : Buffer Range renames Buffer.Head; Tail : Buffer Range renames Buffer.Tail; -- -- begin procedure Add_Element begin

-- -- make sure buffer isn't full
if Head = Tail then
 raise Buffer_Full;
end if;
LIST(Tail) := New Element;
Buffer Length := Buffer Length + 1;
if TaiI = Buffer_Size then
 Tail := 0;
else
 Tail := Tail + 1;
end if;

Page 1884

```
end Add Element ;
```

pragma PAGE; procedure Retrieve Element (Buffer : in out Buffers; Old Element : out Elements) is ------ declaration section --_____ _ _ Buffer Length : Buffer Range renames Buffer.Buffer Length; Head : Buffer Range renames Buffer.Head; LIST : Lists renames Buffer.LIST; Tail : Buffer Range renames Buffer Tail; -- -- begin procedure Retrieve Element __ ____

begin

```
-- make sure don't have an empty buffer
---
      if Head = (Tail-1) or else (Tail = 0 and Head = Buffer Size) then
          raise Buffer Empty;
      end if;
      if Head = Buffer Size then
          Head := 0;
      else
          Head := Head + 1;
      end if;
      Old Element := LIST(Head);
      Buffer Length := Buffer Length - 1;
   end Retrieve Element ;
pragma PAGE;
   function Peek (Buffer : in Buffers) return Elements is
       _____
----
      -- declaration section
      ______
----
      Buffer Length : Buffer Range renames Buffer.Buffer Length;
      Head : Buffer Range renames Buffer.Head;
Tail : Buffer Range renames Buffer.Tail;
LIST : Lists renames Buffer.LIST;
                 : Buffer_Range;
      Spot
     _____
-- -- begin function Peek
__ ______________
   begin
```



```
if Head = (Tail-1) or else (Tail = 0 and Head = Buffer Size) then
         raise Buffer Empty;
      end if;
      if Head = Buffer Size then
        Spot := 0;
      else
        Spot := Head + 1;
      end if;
     return LIST(Spot);
  end Peek ;
pragma PAGE;
   function Buffer Status (Buffer : in Buffers) return Buffer Statuses is
       ------
     -- declaration section
----
     ------
--
     Head : Buffer Range renames Buffer.Head;
     Tail : Buffer Range renames Buffer.Tail;
     Status : Buffer Statuses;
__ _
    -------
-- -- begin function Buffer Status
begin
     if Head = (Tail-1) or else (Tail = 0 and Head = Buffer Size) then
        Status := Empty;
     elsif Head - "ail then
        Status := Full;
     else
        Status := Available;
     end if;
     return Status;
  end Buffer_Status ;
pragma PAGE;
  function Buffer Length (Buffer : in Buffers) return Buffer Range is
  begin
     return Buffer.Buffer_Length;
  end Buffer Length ;
end Bounded_Fifo_Buffer;
```

with Unchecked Deallocation; separate (Abstract Data Structures) package body Unbounded Fifo Buffer is -- -- declaration section __ ____ -- -- this variable is accessed ONLY when setting up the available space list Initial Head : Pointers := new Nodes; Available Space : Buffers := (Current Length => 0, Head => Initial Head. Tail => Initial Head); Available_Length : INTEGER renames Available Space.Current Length; Available Head : Pointers renames Available Space.Head; Available Tail : Pointers renames Available Space.Tail; procedure Free is new Unchecked Deallocation (Object => Nodes, NAME => Pointers); procedure Free Node (Which Node : in out Pointers) renames Free: function Dot Next (Ptr : in Pointers) return Pointers; procedure Set Next (Ptr : in Pointers; Ptr_Dot_Next : in Pointers); package Available Space Operations is new Available Space List Operations (Nodes => Nodes, Pointers => Pointers, Available Length => Available Length, Available Head => Available Head, Available Tail => Available_Tail); function New Node return Pointers renames Available Space Operations.New Node; procedure Save Node (Saved Node : in Pointers) renames Available Space Operations.Save Node; procedure Save List (Saved Head : in Pointers; Saved Tail : in Pointers; Node Count : in POSITIVE) renames Available Space Operations.Save List; pragma PAGE; procedure Initialize Buffer (Buffer : in out Buffers) is _____ -- declaration section -----

```
Current Length : INTEGER renames Buffer.Current Length;
     Head
                   : Pointers renames Buffer.Head;
     Tail
                    : Pointers renames Buffer.Tail;
               -- -- begin procedure Initialize Buffer
begin
     if Current Length = -1 then
        -- handle an uninitialized buffer
----
        Head := New Node;
        Tail := Head;
        Current Length := 0;
     elsif Current Length > 0 then
        -- handle a buffer that has something in it
        Clear Buffer(Buffer => Buffer);
     else
        -- current length = 0 so it is already initialized
        null:
     end if;
  end Initialize Bufter ;
pragma PAGE;
  procedure Clear Buffer (Buffer : in out Buffers) is
       _____
     --- declaration section
---
     _____
--
     Current_Length : INTEGER renames Buffer.Current_Length;
     Head : Pointers renames Buffer.Head;
Tail : Pointers renames Buffer.Tail;
     This Node : Pointers;
-- -- begin procedure Clear Buffer
  begin
     -- make sure this is an initialized buffer
     if Current Length = -1 then
        raise Buffer_Not_Initialized;
```

end if; -- -- placed nodes in the available space list

Save List (Saved Head => Head.Next,

Q

```
Saved Tail => Tail,
               Node Count => Current Length);
     -- reinitialize buffer variables
---
     Current Length := 0;
     Head.Next := null;
     Tail
                 := Head;
  end Clear Buffer ;
pragma PAGE;
  procedure Free Memory is
     _____
-----
     -- declaration section
--
     -----
___
     Node To Be Freed : Pointers;
     ______
-- -- begin procedure Free Memory
__ ___
  begin
     Clear Out Available Space List:
       while Available Head /= Available Tail loop
          Node To Be Freed := Available Head;
          Available Head := Available Head.Next;
          Free Node (Which Node => Node To Be Freed);
       end loop Clear Out Available Space List;
     Available_Length := 0;
  end Free Memory ;
pragma PAGE;
  procedure Add_Element (New_Element : in Elements;
                      Buffer : in out Buffers) is
       _____
--
    -- declaration section
     Current_Length : INTEGER renames Buffer.Current_Length;
    Tail
           : Pointers renames Buffer.Tail;
    New Tail : Pointers;
    -- -- begin procedure Add Element
______
```

Page 1889

begin

```
----
      -- make sure buffer has been initialized
      if Current Length = -1 then
         raise Buffer Not Initialized;
      end if;
     -- now get a node
-----
     New Tail := New Node;
     -- now adjust the buffer
--
     Tail.Next := New_Tail;
     Tail := New Tail;
Tail.Data := New Element;
     Current Length := Current Length + 1;
   end Add Element ;
pragma PAGE;
   procedure Retrieve Element (Buffer : in out Buffers;
                              Old_Element : out Elements) is
--
     ------
     -- declaration section
--
     *-----
---
     Current Length : INTEGER renames Buffer.Current Length;
     Head : Pointers renames Buffer.Head;
     This_Node : Pointers;
-- -- begin procedure Retrieve Element
begin
     -- make sure an element is available
--
     if Current Length = -1 then
        raise Buffer Not Initialized;
     elsif Current Length = 0 then
        raise Buffer Empty;
     end if;
     -- save dummy node in the available space list
___
     This Node := Head;
     Head := Head.Next;
     Save Node (Saved Node => This Node);
     -- retrieve element (its node becomes the new dummy node)
     Old Element := Head.Data;
     -- update buffer status
----
     Current Length := Current Length - 1;
  end Retrieve_Element ;
```

pragma PAGE;

```
function Peek (Buffer : in Buffers) return Elements is
___
     _____
     -- declaration section
--
--
     ------
     Current Length : INTEGER renames Buffer.Current Length;
            : Pointers renames Buffer.Head;
     Head
-- -- begin function Peek
hegin
     -- make sure something is there to look at
---
     if Current Length = -1 then
     raise Buffer Not Initialized;
elsif Current Length = 0 then
       raise Buffer Empty;
     end if;
     return Head.Next.Data;
  end Peek ;
pragma PAGE;
  function Buffer Status (Buffer : in Buffers) return Buffer Statuses is
-----
     -- declaration section
--
     ___
     Current Length : INTEGER renames Buffer.Current Length;
     Status : Buffer Statuses;
-- -- begin function Buffer Status
begin
     if Current_Length = -1 then
       Status := Uninitialized;
    elsif Current Length = 0 then
       Status := Empty;
    else
       Status := Available;
    end if;
    return Status;
  end Buffer Status ;
```



```
pragma PAGE;
   function Buffer Length (Buffer : in Buffers) return NATURAL is
___
      -----
     -- declaration section
----
      ------
___
      Current Length : INTEGER renames Buffer.Current Length;
                         -------
-- -- begin function Buffer Length
____
   begin
     -- make sure the buffer has a length
___
      if Current Length = -1 then
     raise Buffer_Not_Initialized;
      end if;
      return Current_Length;
   end Buffer Length ;
pragma PAGE;
   function Dot_Next (Ptr : in Pointers) return Pointers is
   begin
      return Ptr.Next;
   end Dot Next;
pragma PAGE;
  procedure Set_Next (Ptr : in Pointers;
                       Ptr Dot Next : in Pointers) is
   begin
      Ptr.Next := Ptr Dot Next;
   end Set_Next;
pragma PAGE;
                             --------------
-- begin package Unbounded FIFO Buffer
-- (see header for package body for details)
begin
-- -- set up available space list if one is desired
   if Initial Available Space Size > 0 then
     Add Nodes To Available Space List:
for I in I..Initial_Available_Space_Size loop
            Available_Tail.Next := new Nodes;
            Available Tail := Available Tail.Next;
         end loop Add_Nodes_To_Available_Space_List;
     Available Length := Initial Available Space Size;
  end if;
```

end Unbounded_Fifo_Buffer;

. .

.

```
separate (Abstract Data Structures)
package body Nonblocking Circular Buffer is
pragma PAGE;
  procedure Clear Buffer (Buffer : out Buffers) is
       ------
     -- declaration section
___
     ------
___
     Head: Buffer Range renames Buffer.Head;Tail: Buffer Range renames Buffer.Tail;
     Current Length : Buffer Range renames Buffer.Current Length;
-- -- begin procedure Clear Buffer
 begin
           := 0;
:= 1;
     Head
     Tail
     Current Length := 0;
  end Clear Buffer ;
pragma PAGE;
  procedure Add Element (New Element : in Elements;
                       Buffer : in out Buffers) is
___
     ------
     -- declaration section
     ___
     Head : Buffer Range renames Buffer.Head;
Tail : Buffer Range renames Buffer.Tail;
     Current Length : Buffer Range renames Buffer.Current Length;
           : Lists renames Buffer.LIST;
     LIST
-- -- begin procedure Add Element
```

begin

LIST(Tail) := New_Element;

if Head = Tail then

-- -- buffer was already full and an element was overwritten: therefore, -- -- both head and tail need to be advanced, but Current_Length does -- -- not need to be changed

```
if Tail = Buffer_Size then
   Head := 0;
   Tail := 0;
else
```

Page 1894

```
Head := Head + 1;
Tail := Tail + 1;
end if;
```

else

```
-- buffer was not already full; therefore, the Current Length needs
-- to be increment and only the tail needs to be advanced
```

```
if Tail = Buffer_Size then
   Tail := 0;
else
   Tail := Tail + 1;
end if;
```

```
Current Length := Current Length + 1;
```

end if;

```
end Add Element ;
```

-- -- declaration section

```
Head: Buffer_Range renames Buffer.Head;Tail: Buffer_Range renames Buffer.Tail;Current_Length: Buffer_Range renames Buffer.Current_Length;LIST: Listsrenames Buffer.LIST;
```

```
-- -- begin procedure Retrieve Element
```

begin

```
-- -- make sure there is something there to retrieve
if Current_Length = 0 then
raise Buffer_Empty;
end if;
-- -- advance the head to get to the next element to go out
if Head = Buffer_Size then
Head := 0;
else
Head := Head + 1;
end if;
-- -- now retrieve the element and update the state of the buffer
Old Element := LIST(Head);
```

```
Current_Length := Current_Length - 1;
```

end Retrieve Element ;

```
pragma PAGE;
   function Peek (Buffer : in Buffers) return Elements is
     --
    -- declaration section
___
     -----
            : Buffer Range renames Buffer.Head;
     Head
     Current_Length : Buffer_Range renames Buffer.Current_Length;
     LIST
                 : Lists —
                               renames Buffer.LIST;
     Spot
                 : Buffer Range;
__ _____
-- -- begin function Peek
begin
--
     -- make sure there is something to peek at
     if Current Length = 0 then
      raise Buffer_Empty;
    end if;
     -- determine location of desired element
---
     if Head = Buffer_Size then
        Spot := 0;
     else
        Spot := Head + 1;
     end if;
     -- return requested element
---
     return LIST(Spot);
  end Peek ;
pragma PAGE;
  function Buffer_Status (Buffer : in Buffers) return Buffer Statuses is
___
     --- declaration section
___
     ___
     Current Length : Buffer Range renames Buffer.Current Length;
     Status : Buffer Statuses;
-- -----
-- -- begin function Buffer Status
begin
```

if Current_Length = 0 then
 Status := Empty;
elsif Current_Length = Buffer_Size then

Č

```
Status := Full;
else
Status := Available;
end if;
return Status;
end Buffer_Status ;
pragma PAGE;
function Buffer_Length (Buffer : in Buffers) return Buffer_Range is
begin
return Buffer.Current_Length;
end Buffer_Length ;
end Buffer_Length ;
end Nonblocking_Circular_Buffer;
```

with Unchecked Deallocation; separate (Abstract Data Structures) package body Unbounded Priority Queue is __ ______ -- -- declaration section __ _____ -- -- this pointers is accessed ONLY when setting up the Available Space Initial Head : Pointers := new Nodes; Available_Space : Queues := (Current_Length => 0, Head => Initial Head, Tail => Initial Head); Available Length : INTEGER renames Available Space.Current Length; Available_Head : Pointers renames Available_Space.Head; Available Tail : Pointers renames Available Space.Tail; procedure Free is new Unchecked Deallocation (Object => Nodes, NAME => Pointers); procedure Free Node (Which Node : in out Pointers) renames Free; function Dot Next (Ptr : in Pointers) return Pointers; procedure Set Next (Ptr : in Pointers; Ptr Dot Next : in Pointers); package Available Space Operations is new Available Space List Operations (Nodes => Nodes, Pointers => Pointers, => Available Length, Available Length Available Head Available Tail => Available Head, => Available Tail); function New Node return Pointers renames Available Space Operations.New Node; procedure Save Node (Saved Node : in Pointers) renames Available_Space_Operations.Save_Node; procedure Save_List (Saved_Head : in Pointers; Saved Tail : in Pointers; Node_Count : in POSITIVE) renames Available Space Operations.Save List; pragma PAGE; procedure Initialize (Queue : in out Queues) is ----------- declaration section --_____

```
Current Length : INTEGER renames Queue.Current Length;
      Head
                   : Pointers renames Queue.Head;
      Tail
                   : Pointers renames Queue.Tail;
-- -- begin procedure Initialize
begin
      if Current_Length = -1 then
         -- handle an uninitialized queue
-----
        Head := New Node;
        Tail := Head:
        Current Length := 0;
     elsif Current Length > 0 then
         -- handle a queue that has something in it
_ _
        Clear Queue(Queue => Queue);
     else
        -- current length = 0 so it is already initialized
--
        null;
     end if;
  end Initialize ;
pragma PAGE;
  procedure Clear Queue (Queue : in out Queues) is
          _____
     -- declaration section
---
     ----
     Current Length : INTEGER renames Queue.Current Length;
     Head : Pointers renames Queue.Head;
     Tail
                   : Pointers renames Queue.Tail;
     This Node : Pointers;
-- -- begin procedure Clear Queue
-- -----
  begin
     -- make sure this is an initialized queue
--
     if Current Length = -1 then
```

raise Queue_Not_Initialized;

elsif Current Length > 0 then

.

```
-- placed nodes in the available space list
___
        Save List (Saved Head => Head.Next,
                  Saved Tail => Tail,
                  Node_Count => Current_Length);
        -- reinitialize queue variables
___
        Current Length := 0;
        Head.Next := null;
Tail := Head;
     end if;
  end Clear Queue ;
pragma PAGE;
  procedure Free Memory is
     ___
     -- declaration section
___
     _____
--
     Node To Be Freed : Pointers;
_ _____
-- -- begin procedure Free Memory
begin
     Clear_Out_Available_Space_List:
        while Available Head /= Available Tail loop
           Node To Be Freed := Available Head;
           Available Head := Available Head.Next;
           Free_Node (Which_Node => Node_To_Be_Freed);
        end loop Clear Out Available Space List;
     Available Length := 0;
  end Free Memory ;
pragma PAGE;
  : in out Queues) is
                       Queue
     ----------------
___
___
     -- declaration section
     _____
___
     Current Length : INTEGER renames Queue.Current Length;
     Head : Pointers renames Queue.Head;
Tail : Pointers renames Queue.Tail;
     Before : Pointers;
Here : Pointers;
  ______
```

-

```
Page 1901
```

```
-- -- begin procedure Add Element
__ _____
   begin
      -- make sure queue has been initialized
----
      if Current Length = -1 then
        raise Queue Not Initialized;
     end if;
     -- find the nodes which are to go before and after the new element
___
     Before := Head;
      loop
        exit when (Before = Tail) or else
                   (New Priority > Before.Next.PRIORITY);
        Before := Before.Next;
     end loop;
     -- now get a new node
--
     Here := New_Node;
--
     -- set up the new node
     Here.PRIORITY := New Priority;
     Here.Data := New Element;
Here.Next := Before.Next;
     Before.Next := Here;
     -- readjust the tail, if required
--
     if Before = Tail then
        Tail := Here;
     end if;
---
     -- now adjust the queue
     Current Length := Current Length + 1;
  end Add Element ;
pragma PAGE:
  procedure Retrieve_Element (Queue : in out Queues;
                              Old Element : out Elements) is
      ---
     -- declaration section
     ___
     Current Length : INTEGER renames Queue.Current Length;
     Head : Pointers renames Queue.Head;
     This Node : Pointers;
   -- -- begin procedure Retrieve Element
begin
```

-- make sure an element is available

```
if Current_Length = -1 then
    raise Queue Not Initialized;
elsif Current_Length = 0 then
    raise Queue_Empty;
end if;
```

- -- -- save dummy node in the available space list
 This_Node := Head;
 Head `= Head.Next;
 Save_Node (Saved_Node => This_Node);
- -- -- retrieve element (its node becomes the new dummy node) Old Element := Head.Data;
- -- -- update queue status Current_Length := Current_Length - 1;

end Retrieve Element ;

pragma PAGE;

function Peek (Queue : in Queues) return Elements is

- -- -- declaration section

Current_Length : INTEGER renames Queue.Current_Length; Head : Pointers renames Queue.Head;

- -- -- begin function Peek

begin

```
-- -- make sure something is there to look at
if Current_Length = -1 then
raise Queue_Not_Initialized;
elsif Current_Length = 0 then
raise Queue_Empty;
end if;
return Head.Next.Data;
```

end Peek ;

pragma PAGE;
 function Queue_Status (Queue : in Queues) return Queue_Statuses is

-- -- declaration section

__ ______________________

Current_Length : INTEGER renames Queue.Current_Length; Status : Queue_Statuses;

-- -- begin function Queue Status __ _____ begin if Current Length = -1 then Status := Uninitialized: elsif Current Length = 0 then Status := Empty; else . ز. ا Status := Available; end if; return Status; end Queue Status ; pragma PAGE; function Queue Length (Queue : in Queues) return NATURAL is _____ ------- declaration section _____ ----Current Length : INTEGER renames Queue.Current Length; -- -- begin function Queue Length begin -- make sure the queue has a length --if Current Length = -1 then raise Queue Not Initialized; end if; return Current_Length; end Queue Length ; pragma PAGE; function Dot Next (Ptr : in Pointers) return Pointers is begin return Ptr.Next; end Dot Next; pragma PAGE; : in Pointers; procedure Set_Next (Ptr Ptr_Dot_Next : in Pointers) is begin Ptr.Next := Ptr_Dot_Next; end Set Next; pragma PAGE;

-- begin package Unbounded Priority Queue

-- (see header for package body for details)

begin

-- -- set up available space list if one is desired if Initial Available Space_Size > 0 then

Add_Nodes_To_Available_Space_List:
 for I in I..Initial_Available_Space_Size loop
 Available_Tail.Next := new Nodes;
 Available_Tail := Available_Tail.Next;
 end loop Add_Nodes_To_Available_Space_List;

Available_Length := Initial_Available_Space_Size;

end if;

1

end Unbounded_Priority_Queue;

.
```
separate (Abstract Data Structures)
package body Bounded Stack is
pragma PAGE;
  procedure Clear_Stack (Stack : out Stacks) is
  begin
     Stack.Top := 0;
  end Clear Stack ;
pragma PAGE;
  procedure Add_Element (New_Element : in Elements;
                      Stack : in out Stacks) is
___
     -- declaration section
___
     LIST : Lists
                  renames Stack.LIST;
     Top : Stack Length Range renames Stack. Top;
-- -- begin procedure Add Element
begin
     -- make sure the stack is not already full
---
     if Top = Stack Size then
       raise Stack Full;
     end if;
     -- add element to the stack
___
     Top := Top + 1;
     LIST(Top) := New_Element;
  end Add_Element ;
pragma PAGE;
  procedure Retrieve Element (Stack : in out Stacks;
                          Old_Element : out Elements) is
---
     ------
     -- declaration section
--
___
     LIST : Lists
                           renames Stack.LIST;
    Top : Stack Length Range renames Stack. Top;
-- -- begin procedure Retrieve Element
  _____
```

begin

```
-- make sure there is something in the stack to retrieve
-----
      if Top = 0 then
        raise Stack Empty;
      end if;
     -- retrieve and remove the top element from the stack
     Old Element := LIST(Top);
     Top
               :≡ Top - 1;
   end Retrieve_Element ;
pragma PAGE;
   function Peek (Stack : in Stacks) return Elements is
     --
     -- declaration section
___
     LIST : Lists
                   renames Stack.LIST;
     Top : Stack Length Range renames Stack.Top;
__ ______
-- -- begin function Peek
__ ______
   begin
     -- make sure there is something in the stack
---
     if Top = 0 then
        raise Stack_Empty;
     end if;
     -- return value in top element of the stack
---
     return LIST(Top);
   end Peek ;
pragma PAGE;
   function Stack Status (Stack : in Stacks) return Stack Statuses is
     ---
     -- declaration section
--
     _____
-----
     Тор
         : Stack_Length_Range renames Stack.Top;
     Status : Stack Statuses;
-- -- begin function Stack Status
______
   begin
```

```
if Top = 0 then
   Status := Empty;
elsif Top = Stack_Size then
```

```
Status := Full;
else
Status := Available;
end if;
return Status;
end Stack_Status ;
pragma PAGE;
function Stack_Length (Stack : in Stacks) return Stack_Length_Range is
begin
return Stack.Top;
```

end Stack_Length ;

end Bounded_Stack;

.

with Unchecked_Deallocation; separate (Abstract_Data_Structures) package body Unbounded_Stack is

-- -- declaration section

__ _____

-- -- this pointer is accessed ONLY when setting up the Available Space Initial Head : Pointers := new Nodes;

Available Space : Stacks := (Current Length => 0, => Initial Head. Top Bottom => Initial Head); Available Length : INTEGER renames Available Space.Current Length; Available Top : Pointers renames Available Space.Top; Available Bottom : Pointers renames Available Space.Bottom; procedure Free is new Unchecked Deallocation (Object => Nodes, NAME => Pointers); procedure Free Node (Which Node : in out Pointers) renames Free: function Dot Next (Ptr : in Pointers) return Pointers; procedure Set Next (Ptr : in Pointers; Ptr Dot Next : in Pointers); package Available Space Operations is new Available Space List Operations => Nodes, => Pointers, (Nodes Pointers Available Length => Available Length, Available_Head => Available Top, Available Tail => Available_Bottom); function New Node return Pointers renames Available Space Operations.New Node;

pragma PAGE;

procedure Initialize (Stack : in out Stacks) is

-- declaration section

Page 1908

```
Current_Length : INTEGER renames Stack.Current_Length;
             : Pointers renames Stack.Top;
      Top
      Bottom
                  : Pointers renames Stack.Bottom;
-- -- begin procedure Initialize
begin
      if Current Length = -1 then
         -- handle an uninitialized stack
        Lop:= New Node;Bottom:= Top:
        Current_Length := 0;
     elsif Current_Length > 0 then
         -- handle a stack that has elements in it
___
        Clear Stack (Stack => Stack);
     else
        -- current length = 0, so do nothing
---
        null;
     end if;
   end Initialize ;
pragma PAGE;
   procedure Clear Stack (Stack : in out Stacks) is
     ------
___
     -- declaration section
     ___
     Current Length : INTEGER renames Stack.Current Length;
             : Pointers renames Stack.Top;
: Pointers renames Stack.Bottom;
     TOD
     Bottom
     This_Node : Pointers;
    -- -- begin procedure Clear Stack
begin
     -- make sure stack has been initialized
     if Current_Length = -1 then
        raise Stack_Not_Initialized;
```

--make sure there is something in the stack elsif Current_Length /= 0 then

```
-- placed nodes in the available space list
_
         Save_List (Saved Head => Top.Next,
                   Saved Tail => Bottom,
                   Node Count => Current Length);
         -- reinitialize stack variables
-----
        Top.Next := null;
Bottom := Top;
        Current Length := 0;
      end if;
   end Clear Stack ;
pragma PAGE;
   procedure Free Memory is
      ________________
--
    -- declaration section
     ------
___
     This Node : Pointers;
-- - begin procedure Free Memory
______
  begin
     Deallocate Nodes In Available Space List:
        while Available_Top /= AvaIlable_Bottom loop
           This Node := Available Top;
           Available_Top := Available_Top.Next;
           Free_Node (Which_Node => This_Node);
        end loop Deallocate Nodes In Available Space List;
     Available Length := 0;
     Available Top.Next := null;
  end Free Memory ;
pragma PAGE;
  procedure Add_Element (New_Element : in Elements;
                        Stack : in out Stacks) is
                 _____
     -- declaration section
___
     --
     Current Length : INTEGER renames Stack.Current Length;
            : Pointers renames Stack.Top;
     Тор
```

: Pointers;

Ptr

Page 1910

8

```
CAMP Software Detailed Design Document
         ------
-- -- begin procedure Add Element
       begin
      if Current_Length = -1 then
         raise Stack Not Initialized;
     end if;
     -- get a node and initialize it
     Ptr := New Node;
     Ptr.Data := New Element;
     -- place the node on the stack
     Ptr.Next := Top;
Top := Ptr;
     Current_Length := Current_Length + 1;
  end Add Element ;
pragma PAGE:
  procedure Retrieve Element (Stack : in out Stacks;
```

Current Length : INTEGER renames Stack.Current Length; : Pointers renames Stack.Top;

Old_Element : out Elements) is

```
-- -- begin procedure Retrieve Element
```

-- declaration section

This Node : Pointers;

begin

Тор

-- make sure there is something to retrieve if Current Length = -1 then raise Stack Not Initialized; elsif Current Length = 0 then raise Stack Empty; end if;

-- retrieve data in the top node Old Element := Top.Data;

-- dispose of top node and adjust the stack This Node := Top; := Top.Next; Top Save Node (Saved Node => This Node); Current Length := Current Length - 1;

end Retrieve Element ;

end if;

```
pragma PAGE;
   function Peek (Stack : in Stacks) return Elements is
     _____
     -- declaration section
--
___
     _____
     Current Length : INTEGER renames Stack.Current_Length;
     Тор
            : Pointers renames Stack.Top;
_____
-- -- begin function Peek
__ _____
  begin
     -- make sure there is something to peek at
--
     if Current_Length = -1 then
        raise Stack Not Initialized;
     elsif Current Length = 0 then
        raise Stack Empty;
     end if:
     -- returned desired element
--
     return Top.Data;
  end Peek ;
pragma PAGE;
  function Stack_Status (Stack : in Stacks) return Stack Statuses is
---
     -- declaration section
___
     ------
---
     Current Length : INTEGER renames Stack.Current Length;
     Status
                   : Stack_Statuses;
____
-- -- begin function Stack Status
begin
     if Current Length = -1 then
        Status := Uninitialized;
     elsif Current Length = 0 then
        Status := Empty;
     else
        Status := Available;
```

Page 1912

ĺ

```
Page 1913
```

```
return Status;
```

end Stack Status ;

pragma PAGE;

function Stack Length (Stack : in Stacks) return NATURAL is

-- declaration section

Current_Length : INTEGER renames Stack.Current_Length;

-- -- begin function Stack_Length

begin

```
-- make sure stack has been initialized
___
      if Current Length = -1 then
        raise Stack_Not_Initialized;
      end if;
      return Current Length;
  end Stack Length ;
pragma PAGE;
   function Dot_Next (Ptr : in Pointers) return Pointers is
   begin
     return Ptr.Next;
  end Dot Next;
pragma PAGE;
  procedure Set_Next (Ptr : in Pointers;
                     Ptr Dot Next : in Pointers) is
  begin
     Ptr.Next := Ptr_Dot_Next;
  end Set Next;
pragma PAGE;
-- begin package Unbounded Stack
-- (see header for package body for details)
begin
-- -- set up available space list if one is desired
  if Initial_Available_Space_Size > 0 then
     Add Nodes To Available Space List:
        for I In I..Initial Available Space Size loop
           Available Bottom.Next := new Nodes;
```

Available Bottom := Available Bottom.Next;

end loop Add Nodes To Available Space List;



```
Available_Length := Initial_Available_Space_Size;
```

end if;

end Unbounded_Stack;

٠

.

3.3.8 GENERAL UTILITIES

•

•

(This page intentionally left blank.)

ŀ

б

8

THIS REPORT HAS BEEN DELIMITED AND CLEARED FOR PUBLIC RELEASE UNDER DOD DIRECTIVE 5200.20 AND NO RESTRICTIONS ARE IMPOSED UPON ITS USE AND DISCLOSURE, DISTRIBUTION STATEMENT A APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

3.3.8.1 GENERAL UTILITIES TLCSC P361 (CATALOG #P267-0)

This package provides a group of general utility routines used in a missile system.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.8.1.1 REQUIREMENTS ALLOCATION

This part meets requirement R141.

3.3.8.1.2 LOCAL ENTITIES DESIGN

None.

3.3.8.1.3 INPUT/OUTPUT

None.

3.3.8.1.4 LOCAL DATA

None.

3.3.8.1.5 PROCESS CONTROL Not applicable.

3.3.8.1.6 PROCESSING The following describes the processing performed by this part: package body General_Utilities is end General Utilities;

3.3.8.1.7 UTILIZATION OF OTHER ELEMENTS None.

3.3.8.1.8 LIMITATIONS None.

3.3.8.1.9 LLCSC DESIGN

None.

3.3.8.1.10 UNIT DESIGN

3.3.8.1.10.1 INSTRUCTION SET TEST UNIT DESIGN (CATALOG #P268-0)

This part is a generic function which checks for proper processor operation by executing a function and comparing the result to the expected result. If the expected and derived values match, "True" is returned. The part's generic parameter may be any type, but a Test function must be supplied which matches the parameter defined in the specification.

3.3.8.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets requirement R141.

3.3.8.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.8.1.10.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

	Name		Туре		Description						Ī
	Return_Values	1	private		May be any type. function must retu	The Irn.	type	which	the	included	

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

	Name		Туре		Description	
	Test		function		the function to be tested, it must return a value of Return_Values type.	

FORMAL PARAMETERS:

88

The following table describes this part's formal parameters:

	Name		Туре	1	Mode	Description	I
	Correct_Answer	 	Return_Values		in	The answer which is to be compared to what the function returns.	

3.3.8.1.10.1.4 LOCAL DATA

None.

3.3.8.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.8.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
begin
    return Test = Correct_Answer;
-- -- returns true if function and answer are the same
-- -- false if they are not
    end Instruction_Set_Test;
```

3.3.8.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.8.1.10.1.8 LIMITATIONS

None.

.



.



(This page left intentionally blank.)

.

· .

-

package body General_Utilities is

begin

return Test = Correct_Answer;

-- returns true if function and answer are the same

-- false if they are not

end Instruction_Set_Test;

end General_Utilities;



•

S.

(This page left intentionally blank.)



3.3.8.2 COMMUNICATION_PARTS TLCSC P602 (CATALOG #P691-0)

This package provides a group of communication routines used in a missile system.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.8.2.1 REQUIREMENTS ALLOCATION

This part meets requirement R137.

3.3.8.2.2 LOCAL ENTITIES DESIGN

None.

3.3.8.2.3 INPUT/OUTPUT

None.

3.3.8.2.4 LOCAL DATA

None.

6

3.3.8.2.5 PROCESS CONTROL

Not applicable.

3.3.8.2.6 PROCESSING The following describes the processing performed by this part: package body Communication_Parts is end Communication_Parts;

3.3.8.2.7 UTILIZATION OF OTHER ELEMENTS None.

3.3.8.2.8 LIMITATIONS None.

Page 1924

 \mathcal{C}

3.3.8.2.9 LLCSC DESIGN

3.3.8.2.9.1 UPDATE EXCLUSION PACKAGE DESIGN (CATALOG #P692-0)

This part is a generic package containing a task providing a mechanism for ensuring that data accessed by more than one asymchronous task is properly protected for such accesses. The part's generic parameter can be any type.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.8.2.9.1.1 REQUIREMENTS ALLOCATION

This part meets requirement R137.

3.3.8.2.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.8.2.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

1	Name		Туре		Description	
	Element_Type		private		Allows any type to be protected	

Data objects:

The following table summarizes the generic formal objects required by this part:

1	Name		Туре		Description	Ī
	Initial_Value		Element_Type		Allows the data type to be initialized so that the first time Start_Update_Request is called a constraint error is not raised by some uninitialized value.	

FORMAL PARAMETERS:

The following table describes the formal parameters for the task entries in the task contained in this part.

Task	Name	Mode	Туре	Description
Read_ Update	Read_Request Start_Update_ Request	Output Output	Element_Type Element_Type	Contains the value of the returned data. Contains the value of the returned data.
	Complete_ Update_ Request	Input	Element_Type	Contains the new value of the data to replace the protected data.

3.3.8.2.9.1.4 LOCAL DATA

None.

3.3.8.2.9.1.5 PROCESS CONTROL

Not applicable.

3.3.8.2.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
package body Update_Exclusion is
```

```
task Read Update is
   entry Task Read Request( Requested Data : out Element Type );
   entry Task Start Update Request( Old Data : out Element Type );
   entry Task Complete Update Request( New Data : in Element Type );
end Read Update;
procedure Attempt Read( Requested Data : in out Element Type;
                        Result
                                       : out Rendezvous Flags ) is
begin
   select
      Read Update.Task Read Request( Requested Data );
      Result := Success;
   Else
      Result := Failure;
   end select;
end Attempt Read;
procedure Attempt_Read_Wait( Requested_Data : in out Element_Type;
                             Result
                                            : out Rendezvous Flags ) is
begin
   Read Update.Task Read Request( Requested Data );
   Result := Success;
end Attempt_Read_Wait;
procedure Attempt_Read_Delay( Requested_Data : in out Element_Type;
                              Result
                                             : out Rendezvous Flags;
                              Delay Time
                                             : in DURATION )
                                                                      is
```

```
begin
   Result := Failure;
   select
      Read Update.Task Read Request( Requested Data );
      Result := Success:
   or
      DELAY Delay Time;
   end select;
end Attempt Read Delay;
procedure A.tempt Start Update( Old Data : in out Element Type;
                                New Id : out Rendezvous Ids;
                                         : out Rendezvous Flags) is
                                Result
begin
   select
      Read Update.Task Start Update Request( 01d Data => 01d Data );
      New Id := Id;
      Result := Success;
   else
      Result := Failure:
      New Id := 0;
   end select;
end Attempt Start Update;
procedure Attempt Start Update Wait( Old Data : in out Element Type;
                                     New Id : out Rendezvous Ids;
                                     Result
                                               : out Rendezvous Flags ) is
begin
   Read Update.Task Start Update Request( Old Data => Old Data );
   New Id := Id;
   Result := Success;
end Attempt_Start_Update_Wait;
procedure Attempt Start Update Delay( Old Data : in out Element Type;
                                      New Id : out Rendezvous Ids;
                                              : out Rendezvous Flags;
                                      Result
                                      Time
                                               : in DURATION ) is
begin
   Result := Failure;
   select
      Read Update.Task Start Update Request( Old Data => Old Data );
      New Id := Id;
      Result := Success;
   or
      DELAY Time;
   end select;
end Attempt_Start_Update_Delay;
procedure Attempt Complete Update( New Data : in Element Type;
                                   Passed Id : in Rendezvous Ids;
                                             : out Rendezvous Flags ) is
                                   Result
begin
   if Passed Id = Id then
      select
         Read Update.Task Complete Update Request( New Data );
         Result := Success;
      else
```

Page 1926

```
Result := Failure;
            end select;
         else
            Result := Bad Id;
         end if;
      end Attempt Complete Update;
      task body Read Update is
         Protected Data : Element Type := Initial Value;
      begin
         process continually:
            loop
                select
                   accept Task Read Request (Requested Data : out Element Type)
                   do
                      Requested Data := Protected Data;
                   end Task_Read Request;
               or
                   accept Task_Start_Update_Request (Old_Data : out Element_Type)
                   do
                      Old Data := Protected Data;
                   end Task Start Update Request;
                  accept Task_Complete_Update_Request (New_Data : in Element_Type)
                  do
                      Protected Data := New Data;
                  end Task_Complete_Update_Request;
                  if Id = Rendezvous Ids'LAST then
                      Id := Rendezvous Ids'FIRST + 1;
                  else
                      Id := Rendezvous Ids'SUCC( Id );
                  end if;
               or
                   terminate;
               end select;
            end loop process continually;
      end Read Update;
   end Update Exclusion;
3.3.8.2.9.1.7 UTILIZATION OF OTHER ELEMENTS
None.
```

3.3.8.2.9.1.8 LIMITATIONS

None.

6

.

000

3.3.8.2.9.1.9 LLCSC DESIGN

None.

3.3.8.2.9.1.10 UNIT DESIGN

None.

3.3.8.2.10 UNIT DESIGN

None.



.

```
CAMP Software Detailed Design Document
```

```
package body Communication Parts is
  package body Update Exclusion is
     task Read Update is
        entry Task Read Request( Requested Data : out Element Type );
        entry Task Start Update Request( 0Id Data : out Element Type );
        entry Task Complete Update Request( New Data : in Element Type );
     end Read Update;
     procedure Attempt Read( Requested Data : in out Element Type;
                              Result
                                             : out Rendezvous Flags ) is
     begin
        select
           Read Update.Task Read Request( Requested Data );
           Result := Success;
        else
           Result := Failure;
        end select;
     end Attempt_Read;
     procedure Attempt Read Wait( Requested Data : in out Element Type;
                                                  : out Rendezvous Flags ) is
                                   Result
     begin
        Read Update.Task Read Request( Requested Data );
        Result := Success;
     end Attempt Read Wait;
     procedure Attempt Read_Delay( Requested_Data : in out Element_Type;
                                                   : out Rendezvous Flags;
                                    Result
                                    Delay Time
                                                   : in DURATION )
                                                                            is.
     begin
        Result := Failure;
        select
           Read Update.Task Read Request( Requested Data );
           Result := Success;
        or
           delay Delay Time;
        end select;
     end Attempt Read Delay;
     procedure Attempt Start Update( Old Data : in out Element Type;
                                      New Id : out Rendezvous Ids;
                                      Result
                                               : out Rendezvous Flags) is
     begin
        select
           Read Update.Task Start Update Request( Old Data => Old Data );
           New Id := Id;
           Result := Success;
        else
           Result := Failure;
           New Id := 0;
        end select;
     end Attempt_Start_Update;
     procedure Attempt Start Update Wait( Old Data : in out Element Type;
                                           New Id : out Rendezvous Ids;
```

```
Result
                                               : out Rendezvous Flags ) is
begin
   Read Update.Task Start Update Request( Old Data => Old Data );
   New Id := Id:
   Result := Success;
end Attempt Start Update Wait;
procedure Attempt Start Update Delay( Old Data : in out Element Type;
                                      New Id
                                              : out Rendezvous Ids;
                                      Result : out Rendezvous Flags;
                                              : in DURATION ) is
                                      Time
begin
   Result := Failure;
   select
      Read Update.Task Start Update Request( Old Data => Old Data );
      New Id := Id;
      Result := Success;
   or
      delay Time;
   end select;
end Attempt Start Update Delay;
procedure Attempt Complete Update( New Data : in Element Type;
                                   Passed Id : in Rendezvous Ids;
                                   Result : out Rendezvous Flags ) is
begin
   if Passed Id = Id then
      select
         Read_Update.Task_Complete_Update_Request( New Data );
         Result := Success;
      else
        Result := Failure;
      end select;
   else
     Result := Bad Id;
   end if:
end Attempt Complete Update;
task body Read Update is
  Protected Data : Element Type := Initial Value;
begin
  Process Continually:
     loop
         select
            accept Task Read Request (Requested Data : out Element Type)
            do
               Requested Data := Protected Data;
            end Task_Read_Request;
        or
            accept Task Start Update Request (Old Data : out Element Type)
            do
               Old Data := Protected Data;
            end Task Start Update Request;
```

accept Task_Complete_Update_Request (New_Data : in Element_Type)

end Update_Exclusion;

end Communication_Parts;

813

6

ر ار د



.



(This page left intentionally blank.)

.

.

3.3.9 EQUIPMENT INTERFACES

 $\langle i \rangle$

.

.



.

(This page intentionally left blank.)

3.3.9.1 CLOCK HANDLER TLCSC P634 (CATALOG #P270-0)

This package contains the routines required to maintain an internal clock.

The following routines are provided to manipulate the clock: o Reset clock (effectively zeroes out the clock) o Synchronize clock (effectively sets the clock to the specified time) o Current time (effectively reads the internal clock)

In addition, a Converted Time routine is provided to convert a CALENDAR. TIME to the "local time zone".

An Elapsed Time routine is provided to act as a stopwatch. It returns the elapsed time between successive calls to the function. This function is not affected by resetting or synchronizing the clock.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO46.

3.3.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

This part is a parameterless generic.

3.3.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

1	Name	1	Туре		Value		Description	1
	Reference_Time Time_Last_Called		CALENDAR.TIME CALENDAR.TIME		N/A N/A		Internal reference clock maintained by this part Last time the Elapsed Time function was called	

3.3.9.1.5 PROCESS CONTROL

Not applicable.

3.3.9.1.6 PROCESSING

The following describes the processing performed by this part:

package body Clock Handler is

use CALENDAR;

-- --local declarations

-- -----

Reference Time	:	CALENDAR.TIME	:=	CALENDAR. CLOCK
Time Last Called	:	CALENDAR.TIME	:=	CALENDAR. CLOCK

end Clock Handler;

3.3.9.1.7 UTILIZATION OF OTHER ELEMENTS

The following library units were with'd by the package specification of this part: 1. CALENDAR

UTILIZATION OF EXTERNAL ELEMENTS:

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

	Name		Туре		Source		Description	l
Ι	Clock		Function		Calendar	I	Returns the internal system time	Ī

Data types:

The following table summarizes the external types required by this part:

1	Name		Туре		Source		Description	
	TIME		private		CALENDAR		Implementation-dependent representation of time	
İ	DURATION	İ	fixed	İ	STANDARD	İ	Represents a length of time	İ



88

3.3.9.1.8 LIMITATIONS

The following table describes the exceptions propagated by this part:

	Name	When/Why Raised	
	STANDARD. TIME_ERROR	<pre> Raised by the following routines if a difference in times does not fit within the range of type STANDARD.DURATION:</pre>	

3.3.9.1.9 LLCSC DESIGN

None.

é las

3.3.9.1.10 UNIT DESIGN

3.3.9.1.10.1 CURRENT TIME (FUNCTION BODY) UNIT DESIGN

This function returns the time of the current time of the clock. The current time is the time which has passed since the last time the internal clock was reset or since the time specified when the clock was synchronized.

3.3.9.1.10.1.1 REQUIREMENTS ALLOCATION

This part partially meets requirement CAMP R046.

3.3.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.9.1.10.1.3 INPUT/OUTPUT

None.

3.3.9.1.10.1.4 LOCAL DATA

None.

3.3.9.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.9.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

function Current Time return DURATION is

begin

return CALENDAR.CLOCK - Reference Time;

end Current Time;

3.3.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

The following library units were previously with'd and are visible to this part:

1. Calendar

UTILIZATION OF EXTERNAL ELEMENTS:

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

	Name		Туре		Source		Description	
(Clock	Ι	Function	I	Calendar	1	Returns the internal system time	I

Data types:

The following table summarizes the external types required by this part:

	Name		Туре	Ι	Source	I	Description	Ī
	TIME		private		CALENDAR		Implementation-dependent representation o time	f
İ	DURATION	İ	fixed	İ	STANDARD	İ	Represents a length of time	İ

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Clock_Handler:

Page 1938

	Name		Туре		Value		Description	Ī
	Reference_Time		CALENDAR.TIME		N/A		Internal reference clock maintained by this part	

3.3.9.1.10.1.8 LIMITATIONS

The following table describes the exceptions propagated by this part:

	Name	When/Why Raised	
	STANDARD. TIME_ERROR	Raised if the elapsed time does not fit within the range of type STANDARD.DURATION	

3.3.9.1.10.2 CONVERTED TIME (FUNCTION BODY) UNIT DESIGN

This function converts an input time to a local time (i.e., converts it to the "local time zone"). A local time is defined as the difference between the input time and the internal reference time.

```
3.3.9.1.10.2.1 REQUIREMENTS ALLOCATION
```

This part partially meets requirement CAMP RO46.

3.3.9.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.9.1.10.2.3 INFUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

1	Name	1	Туре		Mode		Description	
1	Clock_Time	1	CALENDAR.TIME		In	1	Time to be coverted to a local time	

3.3.9.1.10.2.4 LOCAL DATA

None.



 $G_{i}^{(1)}$

6
3.3.9.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.9.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

function Converted Time (Clock Time : in CALENDAR.TIME) return DURATION is

begin

return Clock Time - Reference Time;

end Converted Time;

3.3.9.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

The following library units were previously with'd and are visible to this part: 1. Calendar

UTILIZATION OF EXTERNAL ELEMENTS:

Data types:

The following table summarizes the external types required by this part:

Ī	Name	Ι	Туре	Ī	Source		Description	
	TIME		private		CALENDAR		Implementation-dependent representation time	of
İ	DURATION	İ	fixed	İ	STANDARD	İ	Represents a length of time	i

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Clock Handler:

	Name	Туре	Value	Description	
	Reference_Time	CALENDAR.TIME	N/A 	Internal reference clock maintained by this part	

3.3.9.1.10.2.8 LIMITATIONS

The following table describes the exceptions propagated by this part:

	Name	When/Why Raised	
	STANDARD. TIME_ERROR	Raised if the elapsed time does not fit within the range of type STANDARD.DURATION	

3.3.9.1.10.3 RESET CLOCK (PROCEDURE BODY) UNIT DESIGN

This procedure effectively zeroes out the internal clock by setting the internal reference time equal to the system time.

3.3.9.1.10.3.1 REQUIREMENTS ALLOCATION

This part partially meets requirement CAMP R046.

3.3.9.1.10.3.2 LOCAL ENTITIES DESIGN

None.

)

3.3.9.1.10.3.3 INPUT/OUTPUT

None.

3.3.9.1.10.3.4 LOCAL DATA

None.

3.3.9.1.10.3.5 PROCESS CONTROL

Not applicable.

3.3.9.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

procedure Reset Clock is

begin

Reference Time := CALENDAR.CLOCK;

end Reset Clock;

84



The following library units were previously with'd and are visible to this part:

1. Calendar

UTILIZATION OF EXTERNAL ELEMENTS:

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

	Name		Туре		Source		Descri	otior	1			 -
	Clock		Function		Calendar		Returns	the	internal	system	time	 I

Data types:

The following table summarizes the external types required by this part:

	Name	Туре		Source		Description		1
	TIME	privat	e	CALENDAR		Implementation-dependent representation of time	of	
	DURATION	fixed	 	STANDARD	İ	Represents a length of time		İ

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Clock_Handler:

	Name		Туре		Value		Description (
	Reference_Time		CALENDAR.TIME		N/A		Internal reference clock maintained by this part

3.3.9.1.10.3.8 LIMITATIONS

None.

Page 1942

Page 1943

3.3.9.1.10.4 SYNCHRONIZE CLOCK (PROCEDURE BODY) UNIT DESIGN

This procedure effectively sets the internal clock to a user-specified time. It does this by setting the reference time to a system (CALENDAR) time - the desired time. By default, the system time used is CALENDAR.CLOCK by the user may supply his own "system" time.

3.3.9.1.10.4.1 REQUIREMENTS ALLOCATION

This part partially meets requirement CAMP R046.

3.3.9.1.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.9.1.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

I	Name		Туре		Mode	1	Description	
	New_Time Clock_Time		STANDARD. DURATION CALENDAR.TIME		In In		Time to which the internal clock should be set System time	

3.3.9.1.10.4.4 LOCAL DATA

None.

3.3.9.1.10.4.5 PROCESS CONTROL

Not applicable.

3.3.9.1.10.4.6 PROCESSING

The following describes the processing performed by this part:

begin

Reference Time := Clock Time - New Time;

end Synchronize Clock;

3.3.9.1.10.4.7 UTILIZATION OF OTHER ELEMENTS

The following library units were previously with'd and are visible to this part:

1. Calendar

UTILIZATION OF EXTERNAL ELEMENTS:

Data types:

The following table summarizes the external types required by this part:

				. <u>.</u> .		. <u>.</u> .		 	 <u>-</u>
	Name		Туре	1	Source	1	Description		 1

I	TIME	private	CALENDAR	Implementation-dependent representation of
	DURATION	fixed	STANDARD	time Represents a length of time

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Clock Handler:

	Name	Ι	Туре	I	Value		Description	Ī
	Reference_Time		CALENDAR.TIME		N/A		Internal reference clock maintained by this part	

3.3.9.1.10.4.8 LIMITATIONS

None.

3.3.9.1.10.5 ELAPSED TIME (FUNCTION BODY) UNIT DESIGN

This function returns the time since the la : call to this function. The first call to this function will result in the time since the package was elaborated. This function is not affected by calls to Reset_Clock or Synchronize_Clock.

3.3.9.1.10.5.1 REQUIREMENTS ALLOCATION

This part partially meets requirement CAMP RO46.

Page 1945

3.3.9.1.10.5.2 LOCAL ENTITIES DESIGN

None.

6

AHL:

3.3.9.1.10.5.3 INPUT/OUTPUT

None.

3.3.9.1.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

	Name	Туре	Value	Description	
	Answer New_Time	STANDARD. DURATION CALENDAR. TIME	N/A N/A	Amount of time which has elapsed last call to this function System time	since the

3.3.9.1.10.5.5 PROCESS CONTROL

Not applicable.

3.3.9.1.10.5.6 PROCESSING

The following describes the processing performed by this part:

function Elapsed Time return STANDARD.DURATION is

-- -- declaration section

Answer : STANDARD.DURATION; New_Time : CALENDAR.TIME := CALENDAR.CLOCK;

-- -- begin function Elapsed Time

```
-----
```

begin

```
Answer := New_Time - Time_Last_Called;
Time_Last_Called := New_Time;
```

return Answer;

end Elapsed_Time;

3.3.9.1.10.5.7 UTILIZATION OF OTHER ELEMENTS

The following library units were previously with'd and are visible to this part:

1. Calendar

UTILIZATION OF EXTERNAL ELEMENTS:

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

	Name		Туре		Source	1	Description	
	Clock		Function		Calendar		Returns the internal system time	

Data types:

The following table summarizes the external types required by this part:

	Name		Туре		Source		Description	
	TIME		private		CALENDAR		Implementation-dependent representation of time	£
İ	DURATION	İ	fixed	İ	STANDARD	İ	Represents a length of time	İ

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Clock_Handler:

1	Name		Туре		Value		Description	
	Time_Last_Called		CALENDAR.TIME		N/A		Last time the Elapsed Time function was called	-

3.3.9.1.10.5.8 LIMITATIONS

The following table describes the exceptions propagated by this part:

Ċ

Page	1947
------	------

1	Name		When/Why	Raised									
	STANDARD. TIME_ERROR		Raised if type STAN	the elapsed	time)N	does	not	fit	within	the	range	of	



.



(This page left intentionally blank.)

```
package body Clock Handler is
   use CALENDAR;
 -- -- local declarations
__ ____
   Reference_Time : CALENDAR.Time := CALENDAR.Clock;
Time_Last_Called : CALENDAR.Time := CALENDAR.Clock;
pragma PAGE;
   function Current Time return DURATION is
   begin
      return CALENDAR.Clock - Reference_Time;
   end Current Time;
pragma PAGE;
   function Converted Time (Clock Time : in CALENDAR.Time)
                                   return DURATION is
   begin
      return Clock Time - Reference Time;
   end Converted Time;
pragma PAGE;
   procedure Reset Clock is
   begin
      Reference Time := CALENDAR.Clock;
   end Reset Clock;
pragma PAGE;
   procedure Synchronize Clock
                (New Time : in STANDARD.DURATION;
                 Clock Time : in CALENDAR.Time := CALENDAR.Clock) is
   begin
      Reference_Time := Clock_Time - New_Time;
   end Synchronize_Clock;
pragma PAGE;
   function Elapsed Time return STANDARD.DURATION is
      -- declaration section
```

-- -----

.

1983 1983

Answer : STANDARD.DURATION; New_Time : CALENDAR.Time := CALENDAR.Clock;

begin

Answer := New_Time - Time_Last_Called; Time_Last_Called := New_Time;

return Answer;

end Elapsed Time;

end Clock_Handler;

4 (NOT USED)

•

8

٠



•

Q.

(This page intentionally left blank.)

5 (NOT USED)

•

(This page intentionally left blank.)



-

-

This paragraph does not apply to this DDD.

.

....

,

.

(This page intentionally left blank.)



.

SUPPLEMENTARY

INFORMATION

12



DEPARTMENT OF THE AIR FORCE

WRIGHT LABORATORY (AFSC) EGLIN AIR FORCE BASE, FLORIDA, 32542-5434



13 Feb 92

ERRATA AD-B130359

REPLY TO ATTN OF: MINOI

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center ATIN: DTIC/HAR (Mr William Bush) Bldg 5, Cameron Station Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

	Technical Report Number	AD 1	Jumbe	r
1.	88-18-V01-4	adb	120	251
2.	88-18-V01-5	Adb	120	252
3.	88-18-V01-6	Adb	120	253
Д.	88-25-Vol-1	adb	120	309
5.	88-25-Vol-2	Adb	120	310
6.7.8.	88-62-Vol-1	adb	129	568
	88-62-Vol-2	Adb	129	569
	88-62-Vol-3	Adb	129-	570
9.	85-93-Vol-1	adb	102-	654
10.	85-93-Vol-2	Adb	102-	655
11.	85-93-Vol-3	Adb	102-	656
12. 13. 14. 15. 16. 17. 18.	88-18-Vol-1 88-18-Vol-2 88-18-Vol-7 88-18-Vol-8 88-18-Vol-9 88-18-Vol-10 88-18-Vol-11 88-18-Vol-11	ADB ADB ADB ADB ADB ADB ADB	120 120 120 120 120 120 120 120	248 249 254 255 256 257 ¥ 258 259
17.				

2. If you have any questions regarding this request call me at DSN 872-4620.

LYNN S. WARGO Wargo

Chief, Scientific and Technical Information Branch l Atch AFDTC/PA Ltr, dtd 30 Jan 92

ERRATA



DEPARTMENT OF THE AIR FORCE HEADQUARTERS AIR PORCE DEVELOPMENT TEST CENTER (AFSC) EGLIN AIR FORCE BASE, FLORIDA 32542-5000



REPLY TO ATTN OF: PA (Jim Swinson, 882-3931)

30 January 1992

SUBJECT: Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

VIRGINIA N. PRIBYLA, Lt Col, SAF Chief of Public Affairs

ì

AFDIC/PA 92-039