

Earth System Modeling Framework

DRAFT ESMF Requirements

ESMF Joint Specification Team: V. Balaji, Tom Bettge, Byron Boville, Tony Craig, Carlos Cruz, Arlindo da Silva, Cecelia DeLuca, Brian Eaton, Bob Hallberg, Chris Hill, Mark Iredell, Rob Jacob, Phil Jones, Brian Kauffman, Jay Larson, John Michalakes, David Neckels, Jim Rosinski, Shepard Smithline, Max Suarez, Weiyu Yang, Mike Young, Leonid Zaslavsky

17th June 2002

NASA Earth Science Technology Office
Computational Technologies Project
CAN 00-OES-01
<http://www.esmf.ucar.edu>

Contents

1 Synopsis	19
I General Requirements	20
2 Target codes and review team	20
3 Introduction	21
4 Project vision	21
5 Scope	22
6 ESMF superstructure	23
7 ESMF infrastructure	23
7.1 Fields and grids	24
7.1.1 Fields	24
7.1.2 Gridded and observational data	24
7.1.3 Grid operations	24
7.2 Utility infrastructure	25
7.2.1 Communication primitives	25
7.2.2 Generic machine interface	25
7.2.3 I/O	25
7.2.4 Performance profiling	25
7.2.5 Time management	25
7.2.6 Error handling	25
7.3 Future plans	26
8 Organization and conventions of detailed requirements chapters	26
8.1 Requirement attributes	26
9 Framework-wide requirements	28
GR1 Computational requirements	28
GR1.1 Language bindings	28
GR1.1.1 Fortran 90 interface	28
GR1.1.2 C++ interface	28
GR1.1.3 C interface	28
GR1.2 Platforms	28
GR1.2.1 IBM SP	28
GR1.2.2 SGI Origin	29
GR1.2.3 Compaq ES	29
GR1.2.4 PC Linux platforms (including cluster)	29
GR1.2.5 Sun-Solaris	29
GR1.2.6 Vector machines running Unix	29
GR1.3 Performance	29
GR1.4 Precision	29
GR1.5 Runtime configurability	30
GR1.5.1 Configurable decomposition	30
GR1.5.2 Configurable resolution	30

GR1.5.3	Configurable paths and directories	30
GR1.6	Bit-reproducibility	30
GR1.6.1	Parallel bit-reproducibility	31
GR1.6.2	Bit-reproducibility on identical configurations	31
GR1.6.3	Non bit-reproducing fast option	31
GR1.7	Error handling	31
GR1.8	Parallel race-condition error handling	31
GR1.9	Modularity	32
GR1.10	Extensibility	32
GR1.11	Flexibility	32
GR1.12	Documentation	32
GR1.13	Systematic build, test, packaging	33
GR1.14	Compatability with batch execution	33
GR1.15	Compatability with interactive execution	33
GR1.16	Compatability with ensemble methods	33
GR1.17	Compatability with multi-institution, multi-component simulations	33
GR1.18	Ease of adoption	34
GR2	Maintenance and support requirement	34
GR3	Integrated resource monitoring and tracking	34
	References	35
II	Superstructure: Control	36
1	Authors, target codes and review team	36
2	Introduction	36
3	Background	36
3.1	Location	37
3.2	Scope	37
3.3	Summary	37
3.4	Examples	38
4	Control requirements	45
CTL1	The control element	45
CTL1.1	Control as main program	45
CTL1.2	Control as subroutine	45
CTL1.3	SPMD	45
CTL1.4	MPMD	45
CTL1.5	Parallelism	45
CTL1.6	Components within address space	46
CTL1.7	Components outside address space	46
CTL1.8	Customized bindings	46
CTL1.9	Automatic binding of tasks to resources	46
CTL1.10	Write and restore from restart state	46
CTL1.11	Restore and reconfigure	47

CTL2 Computational resource management	47
CTL2.1 Global PE list	47
CTL2.2 Query PE attributes	47
CTL2.2.1 ID for PEs	47
CTL2.2.2 Memory node ID	47
CTL2.2.3 Addressable node ID	48
CTL2.2.4 Processing node ID	48
CTL2.2.5 Other machine attributes	48
CTL2.3 Creating PE lists	48
CTL2.3.1 Exact specification	48
CTL2.3.2 Length specification	48
CTL2.3.3 “Free” specification	49
CTL2.3.4 Memory node affinity	49
CTL2.3.5 Addressable node affinity	49
CTL2.3.6 Processing node affinity	49
CTL2.3.7 Component affinity	49
CTL2.3.8 Compact PE list	50
CTL2.3.9 Spawning of PE lists	50
CTL2.3.10 Complementary PE lists	50
CTL2.3.11 Modify PE list request	50
CTL3 Memory management	50
CTL3.1 Mapped memory region for data sharing	51
CTL4 Component initiation	51
CTL4.1 Instantiation	51
CTL5 Component termination	51
CTL5.1 Termination	51
CTL5.2 Release PE list	51
CTL5.3 Component disappearance	52
CTL6 Component scheduling	52
CTL6.1 By timestep	52
CTL6.2 By time	52
CTL6.3 By alarm	52
CTL7 Cross-component scheduling	52
CTL7.1 Exchange scheduling	53
CTL7.2 Serial execution	53
CTL7.3 Concurrent execution	53
CTL7.4 Forward coupling timestep	53
CTL7.5 Backward coupling timestep	53
CTL7.6 Forward-backward	54
CTL8 Cross-component exchange	54
CTL8.1 Explicit exchange	54
CTL8.2 Implicit exchange	54
CTL9 Cross-component signals	54
CTL9.1 Checkpoint request	55
CTL9.2 Data request	55

CTL10 Syntax	55
CTL10.1 Common syntax across platforms	55
CTL10.2 Performance	55
III Superstructure: Components	56
1 Authors, target codes and review team	56
2 Component background	56
2.1 Location	56
2.2 Scope	56
CGC Component requirements	58
CGC2 Components	58
CGC2.1 Component names	58
CGC2.2 Creation	58
CGC2.2.1 Standard creation	58
CGC2.2.2 Creation based on replication	58
CGC2.2.3 Creation of subcomponents	58
CGC2.3 Interaction with subcomponents	58
CGC2.4 Deletion	59
CGC2.5 Operations	59
CGC2.5.1 Initialize	59
CGC2.5.2 Run	59
CGC2.5.3 Halt	59
CGC2.5.4 Prepare output exchange packets	59
CGC2.5.5 Accept input exchange packets	60
CGC2.5.6 Write and restore from restart	60
CGC2.6 Queries	60
CGC2.6.1 Query name	60
CGC2.6.2 Query layout	60
CGC2.6.3 Query run status	60
CGC2.6.4 Query subcomponent layout	61
CGC2.6.5 Query import state	61
CGC2.6.6 Query export state	61
CGC2.6.7 Query state summary	61
CGC2.7 Query exchange packets	61
CGC2.7.1 Query input datasets	62
CGC2.7.2 Query compute parameters	62
CGC2.7.3 Consolidated query responses	62
CGC3 Application components	62
CGC3.1 Application initialize operation	62
CGC3.2 Queries	63
CGC3.2.1 Query case name	63
CGC3.2.2 Query case date	63

CGC4 Gridded components	63
CGC4.1 Gridded components have one or more associated grids	63
CGC4.2 Creation	63
CGC4.3 Queries	63
CGC4.3.1 Query grids	63
CGC4.3.2 Query default grid	64
CGC5 Coupler components	64
CGC5.1 Coupler run operation	64
CGC5.1.1 Coupling operation limited to two components	64
CGC5.1.2 Unlimited number of coupling operations	64
CGC6 General computational requirements	65
CGC6.1 Validity checking	65
CGC6.2 Compute overhead	65
IV Infrastructure Fields and Grids: Fields	66
1 Authors, target codes and review team	66
2 Background	66
2.1 Location	66
2.2 Scope	66
3 Field summary of requirements	66
4 Field requirements	67
FLD1 Fields	67
FLD1.1 Creation	67
FLD1.1.1 Creation with data allocation	67
FLD1.1.2 Creation with external data	67
FLD1.1.3 Creation without data	67
FLD1.1.4 Creation by indexing an existing field	67
FLD1.1.5 Creation with remap	67
FLD1.1.6 Creation by weighted combination	68
FLD1.2 Local memory layout	68
FLD1.3 Deletion	68
FLD1.4 Attributes	68
FLD1.4.1 Default attributes	68
FLD1.4.2 Recommended attributes	69
FLD1.4.3 Add and delete attributes	69
FLD1.4.4 Copy attributes	69
FLD1.4.5 Collective assignment of attributes	69
FLD1.5 Operations	69
FLD1.5.1 Remap data	69
FLD1.5.2 Return grid	70
FLD1.5.3 Return local memory layout	70
FLD1.5.4 Direct data access	70
FLD1.5.5 Data access via copy	70
FLD1.5.6 Set	70

FLD1.5.7	Write and restore from restart	71
FLD1.6	Queries	71
FLD1.6.1	Query name	71
FLD1.6.2	Query number of dimensions	71
FLD1.6.3	Query attributes	71
FLD1.6.4	Query attributes by name	71
FLD1.6.5	Query number of attributes	71
FLD1.6.6	Query presence of data	72
FLD1.6.7	Query number of local/global cells or gridpoints	72
FLD2	Bundles	72
FLD2.1	Creation	72
FLD2.1.1	Creation using field list	72
FLD2.1.2	Creation by indexing an existing bundle	72
FLD2.1.3	Creation with remap	72
FLD2.2	Local memory layout	73
FLD2.3	Deletion	73
FLD2.4	Operations	73
FLD2.4.1	Remap data	73
FLD2.4.2	Insert and remove field	73
FLD2.4.3	Direct data access	73
FLD2.4.4	Data access via copy	74
FLD2.4.5	Set	74
FLD2.4.6	Return field(s)	74
FLD2.4.7	Return grid	74
FLD2.4.8	Return local memory layout	74
FLD2.4.9	Pack bundle	74
FLD2.4.10	Write and restore from restart	75
FLD2.5	Queries	75
FLD2.5.1	Query bundle name	75
FLD2.5.2	Query field names	75
FLD2.5.3	Query number of fields	75
FLD2.5.4	Query number of local/global cells or gridpoints	75
FLD3	Field and bundle I/O	75
FLD3.1	Write	75
FLD3.2	Set destination	76
FLD3.3	Set write frequency	76
FLD3.4	Write indexed values	76
FLD3.5	Set precision	76
FLD4	General computational requirements	76
FLD4.1	Validity checking	76
V	Infrastructure Fields and Grids: Physical Grids	78
1	Authors, target codes and review team	78

2	Background	78
2.1	Location	78
2.2	Scope	78
2.3	Examples	79
3	Physical grid requirements	80
PG1	Physical locations	80
PG1.1	Horizontal locations	80
PG1.1.1	Horizontal coordinates	80
PG1.1.2	Horizontal locations may be points	80
PG1.1.3	Horizontal locations may be polygonal regions	80
PG1.1.4	Horizontal regions may have central points	80
PG1.1.5	Horizontal regions may be circular	81
PG1.1.6	Paths between grid locations may be specified	81
PG1.2	Vertical locations	81
PG1.2.1	Vertical coordinates	81
PG1.2.2	Vertical locations may be points	82
PG1.2.3	Vertical locations may be regions	82
PG1.2.4	Vertical regions have central points	82
PG1.2.5	Vertical locations may have radii of influence	82
PG1.2.6	Vertical locations may include lopped cells	82
PG2	Location streams	82
PG2.1	Location streams may be created	83
PG2.2	Location streams may be destroyed	83
PG2.3	Location streams may be copied	83
PG2.4	Reading streams	83
PG2.5	Writing streams	83
PG2.6	Background grid	84
PG2.7	Location stream attributes	84
PG2.7.1	Fixed length location streams	84
PG2.7.2	Extensible length location streams	84
PG2.7.3	Global attributes: location stream name	84
PG2.7.4	Location stream registry	84
PG2.7.5	Global attributes: number of dimensions	85
PG2.7.6	Global attributes: dimension names	85
PG2.7.7	Global attributes: dimension units	85
PG2.7.8	Global attributes: text or numeric attributes	85
PG2.7.9	Global attributes: number of elements and number in use	85
PG2.7.10	Global attributes: null element location value	86
PG2.7.11	Elements in stream have similar properties	86
PG2.7.12	Elements include values of locations	86
PG2.7.13	Elements may be copied	86
PG2.7.14	Elements may have attributes	86
PG2.7.15	Location streams may contain null (discarded) elements	87
PG2.7.16	Location streams may be queried for valid elements	87
PG2.8	Location stream methods requiring registries of dependent data	87
PG2.8.1	Registry of data streams	87
PG2.8.2	Extensible location streams may be extended	87
PG2.8.3	Extensible location streams may be shortened	88

PG2.8.4	Extensible length location streams may be converted to fixed length	88
PG2.8.5	Fixed length location streams may be converted to extensible length	88
PG2.8.6	Fixed length streams may have null elements moved to end	88
PG3	Physical grids	88
PG3.1	Reading grids	88
PG3.2	Writing grids	89
PG3.3	Local physical grids may be internally generated	89
PG3.4	Null physical grid creation	89
PG3.5	Physical grid query.	89
PG3.6	Cell specification	89
PG3.7	Refinement	90
PG3.8	Regeneration	90
PG3.9	Distributed grid reference	90
PG3.10	Horizontal coordinate independent of vertical	90
PG3.11	Vertical coordinate potentially dependent on horizontal.	90
PG3.12	Dimension extension	91
PG3.13	Dimension reduction	91
PG3.14	Arbitrary dimensional physical grids	91
PG3.15	1- 2- or 3- dimensional local physical grids	91
PG3.16	Index order	91
PG3.17	Dimension reordering	92
PG3.18	Location index determination	92
PG3.19	Index location determination	92
PG3.20	Horizontal physical grids	92
PG3.20.1	Physical grids map projections	92
PG3.20.2	Unstretched cartesian internal generation	93
PG3.20.3	Latitude-longitude internal generation	93
PG3.20.4	Stand-alone global physical grid generation examples	93
PG3.20.5	Supported topologies	93
PG3.20.6	Local physical grid topology consistency checking	94
PG3.20.7	Areas tile sphere	94
PG3.20.8	Staggered grids	94
PG3.20.9	Available subgrids	94
PG3.20.10	Extensible grid point representations	94
PG3.21	Horizontal functional representations	95
PG3.21.1	Horizontal Fourier grids	95
PG3.21.2	Horizontal spherical harmonics grids	95
PG3.21.3	Mixed physical and Fourier grids	95
PG3.21.4	Extensible horizontal functional representations	95
PG3.22	Vertical functional representations	96
PG3.22.1	Vertical user defined functions	96
PG3.23	Area overlap checking	96
PG3.24	Physical grid attributes	96
PG3.24.1	Physical grid name	96
PG3.24.2	Number of dimensions	96
PG3.24.3	Dimension names	96
PG3.24.4	Dimension lengths	97
PG3.24.5	Dimension attributes and units	97
PG3.24.6	Global attributes	97

PG4	Grid metrics	97
PG4.1	Calculation of metrics	97
PG4.2	Reading metrics	98
PG4.3	MKS metric units	98
PG4.4	Available metrics	98
PG4.5	On-demand metrics	98
PG4.6	Query by name	98
PG4.7	Standard metric naming convention	99
PG4.8	Dimensionality of metrics	99
PG4.9	Available structured horizontal quadrilateral grid metrics	99
PG4.9.1	Cell areas	99
PG4.9.2	Half-edge lengths	99
PG4.9.3	Center-to-edge distances	100
PG4.9.4	Full-edge lengths	100
PG4.9.5	Edge-to-edge distances	100
PG4.9.6	Center-to-corner distances	100
PG4.9.7	Cell orientation	100
PG4.10	Available unstructured horizontal grid metrics	100
PG4.11	Vertical metrics	101
PG4.12	Cell volumes	101
PG4.13	Methods for calculating metrics	101
PG4.13.1	Jacobian metric calculation	101
PG4.13.2	Spline metric calculation	101
PG4.13.3	Distance-based metric calculation	102
PG4.14	Additional metrics	102
PG5	Grid masks	102
PG5.1	Arbitrary number of masks	102
PG5.2	Mask names	102
PG5.3	Category masks	103
PG5.4	Multiplicative masks	103
PG5.5	Mask complement	103
VI	Infrastructure Fields and Grids: Distributed Grids	104
1	Authors, target codes and review team	104
2	Distributed grid background	104
3	Background	104
3.1	Scope	104
3.2	Location	104
3.3	Summary	105
4	Distributed grid requirements	106
DG1	Grid definition	106
DG1.1	Generation of a layout	106
DG1.1.1	Subdivide a layout	106
DG1.2	User-specified layout	106
DG1.3	1D decomposition	106

DG1.3.1	Index order	106
DG1.4	2D decomposition	107
DG1.4.1	Index order	107
DG1.4.2	1D distributed arrays associated with a 2D decomposition	107
DG1.5	3D decomposition	108
DG1.6	Generation of domain decomposition	108
DG1.7	User-specified domain decomposition	108
DG1.8	Domain masks	109
DG1.9	Generation of grid topology	109
DG1.10	Validity of grid topology	109
DG1.11	Periodic boundary conditions	109
DG2	Grid information retrieval	109
DG2.1	Exclusive domain retrieval	109
DG2.1.1	Domain extents	109
DG2.1.2	Domain begin and end indices	110
DG2.1.3	Domain index list	110
DG2.1.4	Maximum domain extent	110
DG2.1.5	Exclusive domain list	110
DG2.2	Local domain retrieval	110
DG2.2.1	Domain extents	110
DG2.2.2	Domain begin and end indices	111
DG2.2.3	Domain index list	111
DG2.2.4	Maximum domain extent	111
DG2.2.5	Index translation for globally non-conformant local domains	111
DG2.2.6	Local domain list	111
DG2.3	Memory domain retrieval	112
DG2.3.1	Domain extents	112
DG2.3.2	Domain begin and end indices	112
DG2.3.3	Domain index list	112
DG2.3.4	Maximum domain extent	112
DG2.3.5	Memory domain list	112
DG2.4	Global domain retrieval	113
DG2.4.1	Domain extents	113
DG2.4.2	Domain begin and end indices	113
DG2.5	Layout retrieval	113
DG2.6	Grid topology retrieval	113
DG2.7	Which PE is a point on?	113
DG2.8	Cross-component queries	114
DG3	Grid relations	114
DG3.1	Equality of global domains	114
DG3.2	Equality of domain decomposition	114
DG3.3	Equality of PE assignment	114
DG4	Halo update	115
DG4.1	Unblocked halo update	115
DG4.2	Blocked halo update	115
DG4.3	Wait for completion	115
DG4.4	Validation and invalidation of halo points	115
DG4.5	Arrays of derived type	116

DG4.6	Adjoint of halo	116
DG5	Data transpose	116
DG5.1	Unblocked data transpose	116
DG5.2	Blocked data transpose	116
DG5.3	Wait for completion	116
DG5.4	Arrays of derived type	117
DG5.5	Adjoint of transpose	117
DG6	Gather	117
DG6.1	Allgather	117
DG6.2	Partial gather	117
DG6.3	Adjoint of gather	118
DG7	Scatter	118
DG7.1	Partial scatter	118
DG7.2	Adjoint of scatter	118
DG8	Broadcast	118
DG8.1	Adjoint of broadcast	119
DG9	Bundling	119
DG9.1	Initiate a bundle	119
DG9.2	Add an array	119
DG9.3	Delete an array	120
DG9.4	Merge bundles	120
DG10	Global reduction operations	120
DG10.1	Integer global sum	120
DG10.2	FP and complex global sum	120
DG10.2.1	FP and complex global sum under a mask	120
DG10.2.2	FP and complex global sum along one axis	121
DG10.2.3	FP and complex bit-reproducible global sum	121
DG10.3	FP and complex global checksum	121
DG10.4	Adjoints of all sums except checksum are required	121
DG10.5	Global maximum of integer or FP data	121
DG10.5.1	Location of global maximum	121
DG10.6	Global minimum of integer or FP data	122
DG10.6.1	Location of global minimum	122
DG11	Blocked and unblocked collectives	122
DG12	Grid staggering	122
DG12.1	AGRID	122
DG12.2	BGRID	122
DG12.3	CGRID	123
DG12.4	DGRID	123
DG12.5	EGRID	123
VII	Requirements for Specific Grid Types	123

DG13 Tripolar grid	123
DG13.1 Vector component reversal	124
DG13.2 Redundancy enforcement	124
DG13.3 Validity of grid	124
DG14 Cubed-sphere grid	124
DG14.1 Vector component interchange	124
DG14.2 Redundancy enforcement	125
DG15 Spectral grid	125
DG15.1 Globalize on one axis	125
DG15.2 Transpose axis of globalization	125
DG16 Exchange grid	125
DG17 Icosahedral grid	125
DG18 Reduced grids	126
DG19 Nested grids	126
DG19.1 Discrete data shift on moving nests	126
DG20 Unstructured grids and ungridded data	126
DG20.1 Grid association	126
DG20.2 Domain decomposition	127
DG20.3 Halos and halo updates	127
DG20.4 Data transpose	127
VIII Infrastructure Fields and Grids: Regridding	128
1 Authors, target codes and review team	128
2 Regrid background	128
2.1 Location	128
2.2 Scope	129
RGRegrid requirements	130
RG2 General regridding requirements	130
RG2.1 Creation	130
RG2.2 Destruction	130
RG2.3 Query	130
RG2.4 Change	130
RG2.5 Reading	131
RG2.6 Writing	131
RG2.7 Support for ESMF grids	131
RG2.8 Multiple fields	131
RG2.8.1 Interface requires only data arrays	131
RG2.8.2 Consistency of field bundles	132
RG2.9 Multiple methods per grid pair	132
RG2.10 Consistency of coordinates	132
RG2.10.1 Consistency of coordinates check	132

RG2.11	Interpolation adjoints	132
RG2.12	Masked regridding	133
RG2.12.1	Mask consistency	133
RG2.13	Independence of field	133
RG2.14	Dependence of field	133
RG3	Regridding algorithms	134
RG3.1	Conservation	134
RG3.1.1	Verification of conservation	134
RG3.2	Monotonicity	134
RG3.3	Higher-order schemes	135
RG3.4	Vector fields in physical space	135
RG3.5	Vector fields in logical space	135
RG3.6	Regridding based on index space	135
RG3.6.1	Index space changes	136
RG3.7	Fourier transforms	136
RG3.7.1	Return types for Fourier modes	136
RG3.7.2	Parallel implementations	136
RG3.8	Legendre transforms	137
RG3.8.1	Data types for Fourier modes	137
RG3.8.2	Parallel implementations	137
RG3.9	Other functional transforms	137
RG3.10	Interpolating from gridded data to ungridded data	137
RG3.11	Interpolating from ungridded data to gridded data	138
RG3.12	User-supplied regridding methods	138
RG4	Other utilities	138
RG4.1	Exchange grid	138
IX	Infrastructure Utilities: Time Management	139
1	Authors, target codes and review team	139
2	Time management background	139
2.1	Location	139
2.2	Scope	139
3	Time management summary of requirements	140
4	Time management abbreviations	140
5	Time management requirements	141
TMG1	Time intervals	141
TMG1.1	Specifying time intervals	141
TMG1.2	Time intervals as return values	142
TMG1.3	Resolution	142
TMG1.4	Range of time intervals	142
TMG1.5	Operations	142
TMG1.5.1	Change value	142
TMG1.5.2	Copy	143

TMG1.5.3 Comparison	143
TMG1.5.4 Increment and decrement	143
TMG1.5.5 Division	143
TMG1.5.6 Subdivision	143
TMG1.5.7 Multiplication	144
TMG1.5.8 Magnitude	144
TMG1.5.9 Return in string format	144
TMG2 Time instants	144
TMG2.1 Units and representation	144
TMG2.2 Consistency with time interval	145
TMG2.3 Supported calendars	145
TMG2.3.1 Gregorian calendar	145
TMG2.3.2 No-leap calendar	145
TMG2.3.3 Julian calendar	145
TMG2.3.4 360-day and generic calendar	145
TMG2.3.5 No calendar option	146
TMG2.4 Operations	146
TMG2.4.1 Change time instant value	146
TMG2.4.2 Copy	146
TMG2.4.3 Comparison	146
TMG2.4.4 Increment or decrement by time interval	146
TMG2.4.5 Increment or decrement by a calendar interval	147
TMG2.4.6 Interval between time instants	147
TMG2.4.7 Return in string format	147
TMG2.5 Queries	147
TMG2.5.1 Standard queries	147
TMG2.5.2 Query day of year	148
TMG2.5.3 Query day of week	148
TMG2.5.4 Query day of month	148
TMG2.5.5 Query middle of month	148
TMG2.5.6 Query julian day	148
TMG2.5.7 Query hardware realtime clock	148
TMG3 Clocks	149
TMG3.1 Clock initialization	149
TMG3.2 Multiple clocks	149
TMG3.3 List of clocks	149
TMG3.4 Operations	149
TMG3.4.1 Advance method	149
TMG3.4.2 Reset timestep interval	149
TMG3.4.3 Change clock current time instant	150
TMG3.4.4 Restore clock state	150
TMG3.4.5 Synchronize with external clock	150
TMG3.5 Queries	150
TMG3.5.1 Query number of timesteps	150
TMG3.5.2 Query timestep interval	150
TMG3.5.3 Query start, stop, reference time	151
TMG3.5.4 Query current or previous time instants	151
TMG3.5.5 Query current or previous simulation times	151
TMG3.5.6 “Is Later” query	151

TMG4	Alarms	151
TMG4.1	Alarm initialization	151
TMG4.2	Multiple alarms per component	152
TMG4.3	List and print of alarms	152
TMG4.4	Alarm states	152
TMG4.5	Ring criteria	152
TMG4.5.1	Ring at time instant	152
TMG4.5.2	Ring at interval	152
TMG4.5.3	Initial ring state	153
TMG4.6	Alarm turn-off	153
TMG4.7	Restore alarm state	153
TMG4.8	Alarm queries	153
TMG5	Accuracy of calculations	153
TMG5.1	Exact increment and decrement	153
TMG5.2	Exact interval calculation	154
TMG5.3	Exact subdivision	154
TMG5.4	Floating point accuracy consistent with time step	154
TMG6	Cross-component clock and alarm queries	154
TMG6.1	Cross-component query	155
TMG6.2	Clock and alarm labels	155
TMG7	General computational requirements	155
TMG7.1	Error handling	155
TMG7.1.1	Check validity	155
TMG7.2	Overloaded arithmetic operators	155
TMG7.3	Automatic memory deallocation	156
TMG7.4	Temporary objects	156
TMG7.5	Thread safety	156
X	Infrastructure Utilities: Communication and Memory Kernels	158
1	Target Codes and Review Team	158
2	Background	158
2.1	Location	158
2.2	Scope	158
2.3	Related Material	159
3	Communication and Memory Kernels Abbreviations	159
CMK1	Basic, portable MPI based transport	160
CMK1.1	A baseline MPI based build must be available	160
CMK2	Basic, portable threads based parallelism	160
References		160
XI	Infrastructure Utilities: Configuration Attributes	161

1	Configuration Attributes Overview	162
2	Target Codes and Review Team	162
3	Introduction	162
4	Background	162
4.1	Location	162
4.2	Scope	162
4.3	Related material	163
5	Configuration Attributes Terms	163
6	Configuration Attributes Abbreviations	163
7	Configuration Attributes Requirements	164
CA1	A human friendly format for parameter specification should be provided	164
CA1.1	Text files specifying parameters should allow comments	164
CA1.2	Text file parsing errors should at the least provide	164
CA1.3	Text file syntax should be backwards compatible with NAMELIST formats.	164
CA2	Attributes can be sub-classified	164
CA3	Attributes can be optional and could be introduced at runtime	165
CA4	A system for defaults and overrides should be supported	165
CA5	A system that is compatible with unique naming should be devised	165
CA6	Components should be able to query the attributes of other components	165
CA7	Components should be able to set attributes to be writable by other components	166
	References	166
XII	Infrastructure Utilities: Performance Profiling	167
1	Authors, target codes and review team	167
2	Background	167
2.1	Location	167
2.2	Scope	167
3	Performance profiling requirements	168
PP1	Code section timing	168
PP1.1	Named timers	168
PP1.1.1	Named timer reset	168
PP1.2	Types of time	168
PP1.2.1	User time	168
PP1.2.2	System time	169
PP1.2.3	Wall clock time	169

PP2	Hardware counters	169
PP3	Process granularity	169
PP3.1	Process level	170
PP3.2	Thread level	170
PP4	Reporting	170
PP4.1	Log output	170
PP4.2	API retrieval	170
PP4.3	Statistics	170
PP4.3.1	Thread statistics	171
PP5	Call deactivation	171
PP5.1	Compile deactivation	171
PP5.2	Runtime deactivation	171
XIII	Infrastructure Utilities: Log	172
1	Authors, target codes and review team	172
2	Background	172
2.1	Location	172
2.2	Scope	172
LG	Log requirements	173
LG2	Interface characteristics	173
LG2.1	Fortran write interface	173
LG2.2	Printf style interface	173
LG3	Log states/levels	173
LG4	Output medium	173
LG5	Process organization	174
LG6	Flush command	174
XIV	Infrastructure Utilities: I/O	175
1	Authors, target codes and review team	175
1.1	I/O architecture	175
1.2	Data models	175
1.3	Metadata. ESMF metadata conventions	176
1.4	Data formats	177
1.5	Parallel I/O	178
1.6	Synchronous and asynchronous IO	178
1.7	Location	179
1.8	Scope	179
2	IO requirements	180

IO1	General IO requirements	180
IO1.1	Establish ESMF metadata conventions	180
IO1.2	Provide automatic generation of metadata	180
IO1.3	Provide generation of companion metadata file	180
IO1.4	Provide generation of GrADS control file	180
IO2	IO requirements for supporting different data formats	181
IO2.1	Reading and writing netCDF files for structured gridded data	181
IO2.2	Reading and writing netCDF files on unstructured grids	181
IO2.3	Reading and writing netCDF files for observational data	181
IO2.4	Reading and writing netCDF files using DODS	181
IO2.5	Reading and writing HDF4 files for structured gridded data	181
IO2.6	Reading and writing HDF4 files for data on unstructured grids	182
IO2.7	Reading and writing HDF4 files for observational data	182
IO2.8	Reading and writing HDF5 files for structured gridded data	182
IO2.9	Reading and writing HDF5 files for data on unstructured grids	182
IO2.10	Reading and writing HDF5 files for observational data	182
IO2.11	Reading and writing HDF-EOS files for structured gridded data	183
IO2.12	Reading and writing HDF-EOS files for data on unstructured grids	183
IO2.13	Reading and writing HDF-EOS files for observational data	183
IO2.14	Reading and writing GRIB files for structured gridded data	183
IO2.15	Reading and writing GRIB files for data on unstructured grids	184
IO2.16	Reading and writing native binary files	184
IO2.17	Reading and writing BUFR files	184
IO3	Parallel I/O requirements	184
IO3.1	Single-threaded IO of distributed data	184
IO3.2	Multi-threaded IO of distributed data to a multiple files	185
IO3.3	Multi-threaded read of distributed data from a single file	185
IO3.4	Multi-threaded write of distributed data to a single file	185
IO3.5	Synchronous and asynchronous IO	185
References		186
XV	Glossary	187
IO1	Glossary	187

1 Synopsis

The following document describes the scope and formal requirements of the Earth System Modeling Framework (ESMF) – a NASA sponsored high-performance computing project for Earth science modeling and data assimilation. The document contains a series of parts. Each part comprises of a series of introductory sections and then specifications of **detailed formal requirements** for that part. The document begins with the *General Requirements* (part **I**). The subsequent parts (**II** to **XIV**) elaborate in more detail on the content identified in the *General Requirements*. A glossary of terms used to describe ESMF requirements and that are used to define the system can be found at the end (part **XV**).

Part I

General Requirements

2 Target codes and review team

Review Date: 20 March, 2002

Target Codes

GFDL-SPEC, BGRID, MOM4
HIM
CAM-EUL, CLM, CCSM-CPL
CAM-FV, PSAS
POP, CICE
WRF
MIT-REG, MIT-CPL, ADJ
NSIPP-ATM, NSIPP-OCN
NCEP-ATM, SSI

Reviewers

Balaji, Smithline
Hallberg
Boville
da Silva, Sawyer
Jones
Michalakes
Hill
Suarez
Iredell

Other Reviewers: DeLuca, Neckels, Larson, Jacob

3 Introduction

The ESMF is a complex and ambitious project. The scope of Earth science modeling is so broad, and the term “framework” so ambiguous, that the ESMF name alone provides little clarification as to what the ESMF actually is. In this document, we set about addressing the first wave of questions that typically accompany an introduction to the ESMF project. What will the ESMF do for the modeling community? What kinds of functionality will the ESMF include? What sorts of components will it couple? Is it associated with a standardization effort and if so which interfaces will be standardized? What happens when the initial funding period is over? It so happens that these questions can be largely addressed by describing three types of “general requirements”: an *overall vision* for the project; a *statement of the scope* of the project; and a *formal listing of requirements* that apply to the whole body of ESMF software. The last of these three can be formulated as tangible properties of the framework that can be validated. The *overall vision* and *statement of the scope* sections are included here as general information that provides background statements of overall vision, project mission, project strategy and project philosophy. The formal validation that the system satisfies the broad statements that the *overall vision* and *statement of the scope* sections contain (sections 4, 5, 6 and 7) occurs through the satisfaction of the detailed requirements given in parts (II - XIV) of this document.

The first type of requirement, itemized in the statement of project vision, is sometimes called a “business requirement.” [9] It describes those objectives, however hazy and unquantifiable, whose attainment will ultimately lead the Earth System Community to assess the ESMF as a success or failure. The second type of requirement, discussed in the statement of project scope, describes the infrastructure requirements that the ESMF will fulfil. Here we describe the functionality of the ESMF, and specify what capabilities the ESMF will *not*¹ include. Finally, we list a set of functional requirements – some quite specific – that apply to all ESMF software. These will be referenced in subsequent class requirements documents.

4 Project vision

These are the broad objectives of the ESMF project:

- Facilitate the exchange of scientific codes so that researchers may more easily take advantage of the wealth of resources that are available in smaller-scale, process modeling and may more easily share experience among diverse large-scale modeling and data assimilation efforts.
- Promote the reuse of standard technical software, the development of which now accounts for a substantial fraction of the software development budgets of large groups. Any center developing or maintaining a large system for NWP, climate or seasonal prediction, data assimilation, or basic research now has to solve very similar software engineering and routine computational problems.
- Focus community resources to deal with architectural changes and the lack of appropriate commodity middleware. The technical parts of the codes that would be dealt with in a common framework are also the most sensitive to architectural changes.
- Present the computer industry with a unified, well-defined and well-documented task for them to address in their software design. The scientific community’s influence with the industry might be much enhanced if exercised jointly by the major modeling centers.
- Share the overhead costs of the software engineering aspects of model development: careful design, complete documentation, user training, and comprehensive testing. These are the efforts that are most easily neglected when corners have to be cut.
- Provide institutional continuity to model and data assimilation development efforts.

¹Items that are *not* included will not be included during this initial development effort, but, may be added in any follow-up development effort.

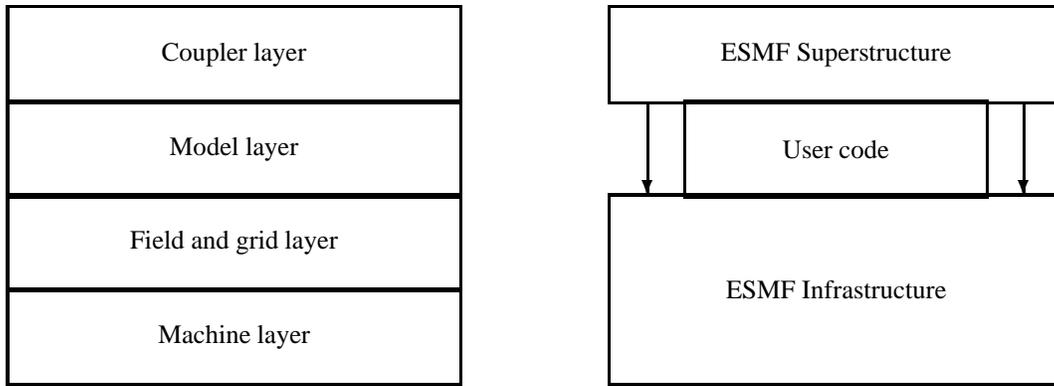


Figure 1: Schematic diagram of ESMF layered architecture.

- Lay the foundation for developing standard classes, components, generic algorithms and data structures that can be used and customized to solve a variety of problems, thereby increasing developer productivity.

These project objectives describe the desired impact of the ESMF on the Earth system modeling community. Although some of these objectives are non-quantitative and difficult to verify, they still must be satisfied to consider the ESMF project truly a success. It is essential to allow them to influence project execution, since it will be possible to satisfy detailed functional requirements but fail to achieve the underlying objectives of our effort.

5 Scope

This section describes the scope of the ESMF software. Specific requirements will be derived from the functionality described here, and presented in more detail in other requirements documents.

The ESMF provides two primary services: to facilitate coupling of Earth system model components, and to support lower-level tasks widely used in Earth Science modeling on high performance computer platforms. We refer to the first of these services as the ESMF coupling *superstructure* and the second as the ESMF utility *infrastructure*.

The ESMF consists of:

- an interface specification for coupling components ² ;
- a complete, portable, high-quality reference implementation ³ accompanied by extensive developer and user documentation;

²The science needs of different users implies a wide variety of continually evolving component interface, for example Aeolian dust deposition is currently being studied as a potentially significant process in climate evolution. Ten years ago almost no atmospheric models or ocean models included these processes, ten years from now there will almost certainly be other feedbacks to incorporate. The component interface specification will provide generic mechanisms, with standard signatures and semantics that support extensible, flexible and self-describing component model interfaces. There are conceptual parallels to such Web projects as the Resource Description Format (<http://www.w3.org/RDF>). Rather than attempting to fix domain-wide vocabularies, by enforcing standardized interfaces, the RDF provides a layer on which such vocabularies may be expressed, and allowed to evolve. ESMF component interfaces likewise will provide the bedrock on which specific component interfaces (e.g an atmospheric model interface) may be expressed. For example the ESMF interface specification will allow an ocean dynamics component to indicate that it can provide sea-surface temperature as an output and will provide a naming mechanism to identify the entities that contain this information. Using the ESMF interface specification atmospheric physics and dynamics components could then indicate that they require a lower-boundary temperature as an input and *attach* their lower-boundary temperature inputs to the ocean model sea-surface temperature output.

³A reference implementation is a widely used approach in deploying open standards like ESMF. A reference implementation is a product (in this case a piece of software) that implements a standard. The ESMF reference implementation will be fully functional and optimized and will be designed and tested for use in real applications. The reference implementation for ESMF will thus be the core framework source code. However, it will also serve to provide other implementors with a standard code to use for validation and to use in bootstrapping development of specialized implementations. Examples of reference implementations of other standards can be found at <http://snad.ncsl.nist.gov/cerberus/>,

- a suite of application examples demonstrating how the ESMF software is used in practice.

The model components that the ESMF supports include atmosphere, ocean, land and sea ice models, and data assimilation systems. Other well-defined high-level functions may also be represented as components in the ESMF superstructure. It is at the level of these large-scale components that the ESMF interface specification applies⁴. Each compliant component will provide a specified set of methods that will enable it to interoperate with other components⁵. For the most part, we expect components to be able to provide the required interface functionality through wrapping of internal data structures and methods.

6 ESMF superstructure

All inter-component communication in the framework occurs through component calls to the superstructure. Here we provide a synopsis of the requirements that the superstructure layer will have to meet. Detailed, tangible requirements for the superstructure, that can be validated, are given in the *Control* and the *Coupler and Gridded Component Interfaces* detailed requirements sections - parts **II** and **III** of this document.

The main functions that ESMF superstructure enables (either directly or through services provided by lower layer functions) are:

1. any merging, interpolation, or regridding of data necessary for communication between components (see parts **III** and **VIII** for detailed requirements relating to this function);
2. transferring data between components (see part **II** for detailed requirements relating to this function);
3. sharing attribute information between components (see part **II** for detailed requirements relating to this function);
4. to coordinate execution, checkpointing and restarting of components (see part **II** for detailed requirements relating to this function).

The superstructure provides a high-level interface that enables applications to be assembled from multiple components. It coordinates the transfer of distributed grid data and ungridded observational data between components. The superstructure will be able to direct the merging of data originating from multiple source grids, and will provide spatial and temporal averaging of data (see parts **II**, **III** and **VIII**).

The ESMF will be capable of operating in multiple modes of execution. components may exist within the same executable, within multiple executables, and in mixed modes. Components may execute serially, concurrently, and in mixed modes (see **CTL1**, part **II**).

Functionality for regridding, interpolation, redistribution and other data communication will be abstracted away from the high-level coupling interface. It is provided by a support layer. This support layer can be used within components to perform efficient parallel grid operations.

7 ESMF infrastructure

The ESMF contains the software necessary to support the data decomposition and communication requirements of the superstructure and individual components. This includes tools for describing field data discretized on a wide variety

<http://www.socks.nec.com/reference/socks5.html>, <http://developer.java.sun.com/developer/onlineTraining/J2EE/Intro/>, <http://www.netlib.org/blas/>, <http://www-unix.mcs.anl.gov/mpi/mpich/workingnote/adi2impl/note.html>. The Netlib BLAS and MPICH examples are probably the most familiar to the Earth science community. Most hardware vendors provide platform specialized variants of these systems. However, these variants are frequently largely derived from the reference codes and are always validated by comparison against the reference code.

⁴The component interface can be applied to smaller components within a model, but, the initial development will not deliver then as part of its milestones.

⁵Extensible, self-describing method interfaces will be used that draw **conceptually** on the ideas employed by systems for internet wide interoperability such as RDF - (see <http://www.w3.org/RDF> and HTTP)

of distributed grids. The ESMF supports distributed data operations in a distributed-memory environment, in a shared memory environment, and in a hybrid computational environment where the platform is a cluster of shared-memory multiprocessors.

7.1 Fields and grids

7.1.1 Fields

The ESMF supports representation, regridding, and other high-level collective operations on vector and scalar fields. A field is represented by metadata, a description of its associated distributed grid, and field data itself. The detailed requirements, with formal validation criteria and relative rankings, for this aspect of ESMF are given in parts **V**, **VIII** and **VI** of this document. Here we provide a brief synopsis of the areas that are covered in that part of the document.

7.1.2 Gridded and observational data

The ESMF supports regridding, interpolation, redistribution, transfer, and merging of data discretized on the following types of grids:

1. logically rectilinear grids: physical co-ordinates may be curvilinear. Logically rectilinear grids include such grids as the tripolar and the cubed sphere;
2. reduced and regional grids;
3. unstructured grids (e.g., land grids);
4. phase space grids (e.g., spectral, Fourier);
5. nested grids;
6. observational data streams (temporal sequences of ungridded data);
7. other grids (e.g icosahedral).

The supplied implementation will include highly-optimized versions of these operations for the commonly-used and representative grid types. The detailed requirements for this aspect of ESMF are given in parts **V** (*Physical Grids* detailed requirements - which deals with what continuous spaces and discrete forms need to be represented) and **VI** (*Distributed Grids* detailed requirements - which deals with parallel representations of these grids) of this document. The detailed requirements include explicit requirements for future extensibility mechanisms to allow support for other grids to be easily introduced.

7.1.3 Grid operations

The ESMF provides interpolation algorithms for supported grids. The detailed requirements, with formal validation criteria and relative rankings, for this aspect of ESMF are given in part **VIII**. Here we provide a brief synopsis of the areas that are covered in that part of the document. All interpolation algorithms included in ESMF are *linear* in the data and will be accompanied by the associated adjoint. Adaptive procedures are not planned at this stage. Thus, reconstruction of gridded data from observations (analysis) will not be a part of framework and will rather be provided by data assimilation components.

The ESMF provides first-order and higher-order interpolation methods. The ESMF supports conservative remapping/interpolation between any two grids. Non-conservative methods will also be supported.

Dynamic load balancing will be provided for standard decompositions, and general dynamic load-balancing tools for specialized decompositions.

7.2 Utility infrastructure

The ESMF includes general purpose utility routines for use by both the ESMF coupler and application codes. The detailed requirements, with formal validation criteria and relative rankings, for this aspect of ESMF are given in parts **IX - XIV** of this document. Here we provide a brief synopsis of the areas that are covered in that part of the document. These utilities, described in parts **IX** to **XIV**, include but are not limited to:

1. communication primitives (see part **X**);
2. a generic machine interface (see part **X**);
3. I/O utilities (see part **XIV**);
4. general message logging (see part **XIII**);
5. mechanisms for publishing and querying component attributes (see part **XI**);⁶
6. performance profiling (see part **XII**);
7. time management (see part **IX**); and
8. error handling (see GR3 and part **XIII**).

7.2.1 Communication primitives

ESMF will offer a standard interface covering inter-processor communication and shared address space primitives, covering point-to-point communication, global reduction operations, etc. An implementation in MPI will be provided.

7.2.2 Generic machine interface

ESMF will offer a generic interface to hardware, O/S and system library primitives.

7.2.3 I/O

The ESMF I/O utilities will include generic interfaces providing I/O in a variety of data formats including netCDF, HDF, binary, GRIB, and BUFR. As much as possible, the utilities will offer scalable, parallel I/O, performance.

7.2.4 Performance profiling

A performance profiling utility will enable application developers to instrument code segments with timers. The profiling utility will also offer access to hardware statistics by offering an interface to a package such as PAPI or PCL.

7.2.5 Time management

The ESMF will include a library for performing routine calculations with dates and time intervals. For higher level model time management, the ESMF will include clocks for advancing and reporting model time and retaining model integration information, and alarms for initiating both unique and periodic events.

7.2.6 Error handling

The ESMF will offer comprehensive and integrated error handling services. A mechanism to announce and query component failure will be provided.

⁶Like many other elements under *Utility infrastructure*, the querying and publishing functions will be fundamental to the superstructure control and coupling areas. Isolating the requirements here does not imply a particular system design organization or ranking.

7.3 Future plans

The following are capabilities that might conceivably have been included in ESMF, but will not be developed under CAN 00-OES-01 funding:

1. a Graphical User Interface;
2. mathematical libraries for operations other than regridding and interpolation;
3. standard scientific modules, such as a library for calculating orbital parameters;
4. a database of intra-model components, such as convection schemes;
5. a database for storing experiments and related information;
6. a mechanism for job submission (e.g Globus);
7. optimization for additional model grids.

If additional resources are made available, the ESMF project foresees extending the initial framework. Each of the features listed above is a possible addition.

8 Organization and conventions of detailed requirements chapters

Requirements documents shall be organized so that specific requirements are listed under a titled topic. For example, under the title **Multi-platform support**, with the description *The ESMF software must run on a number of high-performance computing platforms*, the ability to run on an SGI Origin, IBM SP, Compaq ES40, etc. would be listed as individual and separately numbered requirements.

Templates for requirements documents are available in the ESMF document template set. The ESMF Software Developers Guide describes all the documents that will be produced. The following extract reproduces the text from the Developers guide that describes the attributes of detailed requirements.

8.1 Requirement attributes

Each specific requirement possesses the following attributes: priority, source, verification, status, and notes, the last of which is optional. These are typical for requirements analysis [9]. We'll now look at them in more detail.

Priority The purpose of the priority attribute is to associate each requirement with the milestones and longer term project goals that it satisfies. Each requirement is assigned a number from 1-3, with values defined as follows:

1. This capability is directly required for a milestone OR Half or more of the JMC applications that could use the utility or class in which this capability is embedded would require this capability in order to maintain their existing functionality;
2. Less than half of the JMC applications that could use the utility or class in which this capability is embedded would require this capability in order to maintain their existing functionality.
3. This capability is desired in order to extend the existing functionality of one or more JMC codes.

If some capability merits additional explanation to describe its priority, those preparing requirements are encouraged to elaborate.

Source The source attribute traces each capability to the applications to which it applies. In addition to applications particular people or organizations may be noted. This attribute helps to identify those that can provide further information and who may also be potential testers and users. It prevents the inclusion of features that have little likelihood of being used.

Verification The verification attribute specifies an objective and quantitative strategy for assessing whether a requirement is satisfied. Typical values include *code inspection*, *unit test* and *system test*. Some capabilities may require the preparation of special data sets.

Status Throughout the course of this project it will be useful for us to track what has been accomplished and to archive ideas for extensions and improvements. The status attribute identifies each capability as:

- **proposed**; this indicates an item that has been accepted as useful, but that is not scheduled for implementation.
- **approved-1**; this indicates an item approved for implementing as part of the 1st code release at Milestone F
- **approved-2**; this indicates an item approved for implementing as part of the 2nd code release at Milestone G
- **implemented**
- **verified**
- **rejected**; this indicates an item that has been actively rejected by the review team.

Whether the capability exists in other packages or models is also helpful to note.

Notes This is a catch-all for additional information such as background, references, related design and implementation issues, risk factors, and so on.

9 Framework-wide requirements

Here we describe the requirements that apply to the whole body of ESMF software. We adopt the standard requirements format described in the ESMF Software Developer's Guide. The abbreviation used to identify these General Requirements is **GR**. The requirements are drawn from the applications and user needs from ESMF project participants [1], from community feedback and from background analysis carried out by the NASA computational technologies group [10].

GR1 Computational requirements

GR1.1 Language bindings

ESMF software shall support the following language bindings:

GR1.1.1 Fortran 90 interface

Priority: 1.

Source: All codes except GFDL-HIM require F90 interfaces.

Status: Approved-1.

Verification: Interface inspection (verification checklist will include array dimension information and optional argument handling).

GR1.1.2 C++ interface

Priority: 1.

Source: GFDL-HIM.

Status: Approved-1.

Verification: Interface inspection (verification checklist will include array dimension information and optional argument handling).

Notes: Handling of Try/Catch exception handling and error signal propagation over the C/C++ Fortran boundary will need to be addressed in the preliminary design and implementation prototyping.

GR1.1.3 C interface

Priority: none

Source: GFDL-HIM.

Status: Rejected.

Notes: There will be a single interface serving C and C++ applications. This C/C++ interface will be sufficient so the C only interface has been rejected.

GR1.2 Platforms

The ESMF shall operate on the following platforms:

GR1.2.1 IBM SP

Priority: 1-approved.

Source: NCAR, NCEP.

Status: Approved-1.

Verification: Unit test and system test.

GR1.2.2 SGI Origin

Priority: 1-approved.

Source: GFDL, LANL, NCAR, DAO. May be part of ESMF testbed system.

Status: Approved-1.

Verification: Unit test and system test.

GR1.2.3 Compaq ES

Priority: 1.

Source: NSIPP, DAO. May be part of ESMF testbed system.

Status: Approved-1.

Verification: Unit test and system test.

GR1.2.4 PC Linux platforms (including cluster)

Priority: 1.

Source: MIT. Required for optional cluster milestones.

Status: Approved-1.

Verification: Unit test and system test.

GR1.2.5 Sun-Solaris

Priority: 1.

Source: NCAR.

Status: Approved-1.

Verification: Unit test and system test.

GR1.2.6 Vector machines running Unix

Priority: 2.

Source: NCAR.

Status: Approved-2.

Verification: Unit test and system test.

GR1.3 Performance

Fully compliant adoption of the framework shall not increase the execution time of the milestone G codes, which are maintained and developed outside the framework throughout the project, by more than 10% on scalar architectures within their scalable range.

Priority: 1.

Source: Required for Milestone G.

Status: Approved-2.

Verification: Demonstrate with NCEP analysis and MITgcm ocean.

GR1.4 Precision

ESMF will provide methods for applications using default integers and floating-point numbers at either 32-bit or 64-bit precision.

Priority: 1.

Source: DAO, NCEP.

Status: Approved-1.

Verification: Interface inspection (verification checklist will include array dimension information and optional argument handling) , code inspection, unit test, system test.

Notes:

1. While it is technically possible through overloading to allow argument lists to contain arbitrary combinations of 32-bit and 64-bit quantities, this Requirement only calls for a uniform application-wide word length for integers and FP numbers to be supported.
2. There may be individual operations within ESMF that explicitly call for transformations between 64-bit and 32-bit quantities. Those will appear as individual requirements.

GR1.5 Runtime configurability

The ESMF shall allow the user to set certain application parameters at runtime.

Notes: There are many runtime configurable parameters that need to be set for codes operating as framework components. The sub-requiremente listed here GR1.5.1 - GR1.5.3 are some, but not all of these.

GR1.5.1 Configurable decomposition

ESMF shall allow domain decomposition and assignment of domains to processors, nodes, and/or threads to be configurable at runtime. The choice of parallelization mechanism (message passing/multithreading/combo) must be configurable at runtime.

Priority: 1.

Source: CAM-EUL.

Status: Approved-1.

Verification: System test.

GR1.5.2 Configurable resolution

ESMF shall allow model resolution to be configurable at runtime.

Priority: 1.

Source: all codes.

Status: Approved-1.

Verification: System test.

GR1.5.3 Configurable paths and directories

ESMF shall allow input and output paths and directories to be specified at runtime.

Priority: 1.

Source: all codes.

Status: Approved-1.

Verification: System test.

GR1.6 Bit-reproducibility

Bitwise identical results between two runs is an important feature for deterministic execution, as well as for debugging and maintenance. This is not possible to enforce across platforms, at different levels of compiler optimization, or even different versions of the same compiler. Within these constraints, ESMF shall provide varied levels of bit-reproducibility:

GR1.6.1 Parallel bit-reproducibility

An execution mode will be provided which allows an ESMF-based model to return bitwise identical results from the same executable at different processor counts, or different parallelism options. This execution mode is not subject to the performance requirement of Section GR1.3.

Priority: 1.
Source: all JMC codes.
Status: Approved-1.
Verification: System test.

GR1.6.2 Bit-reproducibility on identical configurations

The *default* execution mode will allow an ESMF-based model to return bitwise identical results from the same executable at the same processor count, with the same parallelism options. This mode *is* subject to the performance requirement of Section GR1.3.

Priority: 1.
Source: all codes.
Status: Approved-1.
Verification: System test.

GR1.6.3 Non bit-reproducing fast option

If a performance enhancement may be had by waiving the default requirement above, this will be made available in a special execution mode. In this mode, even two runs at the same processor count, with the same parallelism options, may not return bitwise identical results.

Priority: 2.
Source: NCEP, HIM.
Status: Approved-2.
Verification: System test.

GR1.7 Error handling

The ESMF shall be instrumented for error handling consistently across the framework. The user shall receive ample information on errors including comprehensive error logging.

Priority: 1.
Source: Required by all codes.
Status: Approved-1.
Verification: Interface inspection (verification checklist will include array dimension information and optional argument handling), code inspection, unit test, system test. **Notes** Components will be required to use ESMF error handling utilities for fatal errors to be classified as *well-behaved* ESMF components.

GR1.8 Parallel race-condition error handling

Error handling code to detect race-conditions and other problems associated with parallelism shall be included.

Priority: 1.
Source: Required by all codes.
Status: Approved-1.
Verification: System test. **Notes** This could be achieved through assertions and ordering/bookeeping counters on messaging on communication operations.

GR1.9 Modularity

Layers of the framework will be designed to adapt to restructuring of other parts of the framework and user-supplied components. For example, the coupling layers should be able to adapt to different implementations and data structures of component models. A modular design and implementation approach must be used that ensures parts of the framework are independent of one another and can be developed concurrently.

Priority: 1.

Source: Required for interoperability Milestones I and J.

Status: Approved-1.

Verification: Interoperability experiments and incremental adoption demonstrations by deployment teams.

Notes: Good encapsulation, modular design and clean interfaces are critical to the maintenance of models built with the framework. Framework classes, modules and algorithms must be easy to adapt to take advantage of new techniques that are developed by the Earth science community and added to the framework.

GR1.10 Extensibility

The ESMF framework must allow extensions both for core support capabilities and through plug-in components. Framework classes, modules and algorithms must be easily extensible to allow additional functionality to be added for supporting new components like atmospheric chemistry, carbon cycle, etc...

Priority: 1.

Source: All codes

Status: Approved-2.

Verification: Milestone demonstrations of migration of more than twelve major applications and six styles of gridding.

Notes: It is important that appropriate abstract interfaces and polymorphic methods are used to ensure adding new methods, grids, algorithms and data structures can be easily done. This will facilitate building new types of models that are not currently available.

GR1.11 Flexibility

The ESMF framework will be flexible and will be suited to a wide range of Earth science applications and a broad span of target hardware. Framework classes, modules and algorithms must be flexible and provide options to support both efficiency and accuracy.

Priority: 1.

Source: All codes

Status: Approved-1.

Verification: Milestone demonstrations of migration of more than twelve major applications on several different platforms covering a wide variety of science scenarios that span research to operational forecasting.

GR1.12 Documentation

An exhaustive collection of documentation to support ongoing evolution of ESMF is required. The framework must be fully documented, easily understandable and well supported.

Priority: 1.

Source: All codes

Status: Approved-1.

Verification: Compliance with software developers guide. **Notes:** The Software Developers Guide specifies a suite of documents that will be produced.

GR1.13 Systematic build, test, packaging

A systematic scheme for disseminating stable, versioned software and for gathering and tracking defects is required. The framework should be delivered to developers using the standard packaging utility for the supported platforms.

Priority: 1.

Source: All codes

Status: Approved-1.

Verification: Compliance with software developers guide. **Notes:** The Software Developers Guide specifies a versioning schemes that will be used, release schedules and mechanisms for monitoring and managing product quality.

GR1.14 Compatability with batch execution

Priority: 1.

Source: All codes

Status: Approved-1.

Verification: Milestone demonstrations.

Notes: In this mode, all input and output data are read and stored directly from file. This will be the standard mode of operations used by many ESMF applications.

GR1.15 Compatability with interactive execution

Priority: 1.

Source: All codes

Status: Approved-1.

Verification: Milestone demonstrations.

Notes This mode allows rapid testing of new features and debugging. A trace mode which allows program execution to be logges in detail, but at the expense of reduved performance, is useful in this mode. Future extensions could include elements of computatinal steering in which program progress can be monitored and directed interactively.

GR1.16 Compatability with ensemble methods

Priority: 1.

Source: All codes

Status: Approved-1.

Verification: Milestone demonstrations.

Notes: Ensemble methods require multiple instances of the same base component, generally with slight parametric variations. The ability to spawn numerous ensembles and then collect together results for automated processing, such as covariance calculation, is a basic requirement. A general mix of concurrent and sequential ensemble member execution must be supported.

GR1.17 Compatability with multi-institution, multi-component simulations

Priority: 1.

Source: All codes

Status: Approved-1.

Verification: Milestone demonstrations.

Notes: Some form of component versioning, labeling and naming is required that enables experiment configuration logs to include information of which components were used.

GR1.18 Ease of adoption

It must be straightforward to integrate ESMF into an application that is reasonably modular. Adapting an existing modular application to use the frameworks coupling services shall require no more than 2% modification of the applications source code.

Priority: 1.

Source: GFDL, NSIPP.

Status: Approved-1.

Verification: Line count.

GR2 Maintenance and support requirement

The ESMF must be maintained as a long-term commitment by at least one institution. This maintenance must extend beyond adaptation to the computational environment, and must include an ongoing research component dedicated to increasing the performance, flexibility and functionality of the software.

Priority: 1.

Source: All participants.

Status: Approved-1.

Verification: Publicly documented commitment, line item in projected budgets.

GR3 Integrated resource monitoring and tracking

The ESMF shall be instrumented to allow resource utilization to be tracked and monitored.

Priority: 1.

Source: Required by all codes.

Status: Approved-1.

Verification: Unit test, system test.

Notes

References

- [1] Earth system modeling framework. <http://www.esmf.ucar.edu>, 2002.
- [2] A. Arakawa. Computational design for long-term numerical integration of the equations of atmospheric motion. *J. Comp. Phys.*, 1:119–143, 1966.
- [3] V. Balaji. Parallel numerical kernels for climate models. In ECMWF, editor, *ECMWF Teracomputing Workshop*. World Scientific Press, 2001.
- [4] R. Heikes and D. A. Randall. Numerical Integration of the Shallow-water Equations of a Twisted Icosahedral Grid. Part I: Basic Design and Results of Tests. *Monthly Weather Review*, 123:1862–1880, 1995.
- [5] P.W. Jones. First- and second-order conservative remapping schemes for grids in spherical coordinates. *Monthly Weath. Rev.*, 127:2204–2210, 1999.
- [6] D. Majewski, D. Liermann, P. Prohl, B. Ritter, M. Buchhold, T. Hanisch, G. Paul, and W. Wergen. The Operational Gblal Icosahedral-Hexagonal Gridpoint Model GME: Description and High-Resolution Tests. *Monthly Weather Review*, 130:319–338, 2002.
- [7] Ross J. Murray. Explicit generation of orthogonal grids for ocean models. *J. Comp. Phys.*, 126:251 – 273, 1996.

- [8] M. Rancic, R.J. Purser, and F. Mesinger. A global shallow-water model using an expanded spherical cube: Gnomonic versus conformal coordinates. *Quart. J. Roy. Meteor. Soc.*, 122:959 – 982, 1996.
- [9] Wiegers, K. E. *Software Requirements*. Microsoft Press, 1999.
- [10] Womack, B., Higgins, G. *Software Engineering Support of the Third Round of Scientific Grand Challenge Investigations. General Requirements Analysis*. NASA/CT Internal Document, 2002.

Part II

Superstructure: Control

1 Authors, target codes and review team

Authors: V. Balaji and Chris Hill

Review Date: 25 April, 2002

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CCSM-CPL	Craig, Kauffman
CAM-EUL, CLM	Boville
CAM-FV, PSAS	da Silva, Sawyer
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell, Young

Other Reviewers: DeLuca, Neckels, Larson, Jacob

2 Introduction

3 Background

The ESMF is a software framework for the construction of applications by coupling modeling and data processing components that are executing on shared- and distributed-memory scalable systems. The components may themselves be parallel. Consequently, a single ESMF application may run on a variety of platforms, perhaps including components coupling across different platforms to form a single heterogeneous *application*.

The construction of such applications involves harnessing of system resources (processors, memory, CPU time, I/O channels, etc.) from operating systems. While individual components may independently manage such resources themselves, it will be necessary to have a functional layer that provides a means of coordinating resource utilization and that acts as a conduit between components. The implementation of this function could range from an extremely general Globus-like approach, to something more tightly focused on the needs of a small community. Begin with the community-wide consultative process of writing this requirements document, we anticipate arriving at an appropriate level of generality for this function. Within ESMF the function we are defining is referred to as the **Control** layer.

Connecting between components also requires a means of data transfer. This involves the actual data (e.g surface fluxes) that is to be shared between components and attributes of components that may be required by others (e.g. metadata describing datatypes a particular component is capable of providing, frequency of availability of such data); and means of signalling between components (e.g to organize a shutdown). There are many possible mechanisms by which data transfers could be expressed and many mechanisms by which data can be transported. The **Control** layer will play an integral role in determining how data from the name space of one component is made available in the name space of another component.

These functions necessarily reside in a layer outside the purview of individual components. The **Control** layer described in the document forms part of the ESMF superstructure to provide a layer of inter-component coordination and connectivity.

3.1 Location

This layer is a major element of the Superstructure. The **Control** layer will be involved in setting up processor subsets (“pelists”) that will be used by various layers in the ESMF Infrastructure (**DistGrid** and **CommKernels**). In conjunction with the **Coupler** layer, it is responsible for setting up and executing data and signal exchange between components.

3.2 Scope

The scope of **Control** is to manage system resources for a multi-component ESMF application and co-ordinate execution and information exchange between components. It is not intended to take over the duties of an operating system scheduler, but rather performs a subset of those operations within the context of a single application. Also, given that OS schedulers vary widely in the kinds of user control they provide, **Control** will allow components to have a systematic way of requesting resources.

As far as possible the **Control** layer will be unaware of the content of components. As such, the detailed logic within a component will be treated as a black box and the **Control** layer will only deal with mapping inputs and outputs between components. In other words, the data connection elements of the **Control** layer are primarily concerned with expressing that certain entities in the namespace of one component map to counterpart entities in the namespace of another component, the content of the entities if of no interest to the **Control** layer. Similarly, the data transport elements of the **Control** layer are primarily concerned with any required movement of data between entities in different name spaces, again with only minimal concern about the actual data being transported. As a result the **Control** layer will not understand that a particular component is a land model and that another component is an ocean model.

In a related area, the sequencing of operations in **Control** layer will be based on a simple imperative structure that will assume that components to be used are known and located ahead of time (**Chris, will you always know this ahead of time in MPMD?**) and that their set of possible inputs is also known outputs. Functions such as dynamic discovery of components with specific capabilities (for example “seeking out” an ocean model that can simulate a particular period at a particular resolution) is not within the scope of the **Control** layer.

It is possible to imagine layering tools such as discovery and component identification tools on top of the basic **Control** layer within ESMF. However, such tools fall outside the scope of the core ESMF development.

3.3 Summary

The Control element is the main program invoked by all programs participating in an ESMF application. Both SPMD and MPMD operations are supported. It is responsible for managing the computational resources assigned to the job, and apportioning them among components. The Control element itself will be able to save its state, and restore it from a file. It will be able to restart on a set of assigned resources different from the saved one.

Components will be able to make requests in the form of a list of computational resources (PEs and memory), and must be capable of working with assigned resources. Components may themselves use shared or distributed memory. Components are free to use the assigned resources as desired; but may also accept hints from the Control element about how to use them.

The Control element also orchestrates data exchanges between components. Components may share data using shared or distributed memory. Data exchanges can be scheduled using Time Manager semantics. Exchanges may be explicit or implicit. The coupling timestepping between components may be forward, backward, or forward-backward.

Components will have a standard interface to define their desired inputs and expected outputs to and from other components. The Control element is responsible for validating the exchanges.

Original text from Boulder meeting, 22 Feb 2002:

parallel initialization and termination, checkpoint management, handshaking between multiple executables, assignment of components to processors, concurrent and serial scheduling options for components, interfaces for data transfer between components, definition of standard external interfaces that components are required to provide.

ESMF “Hello Worlds”

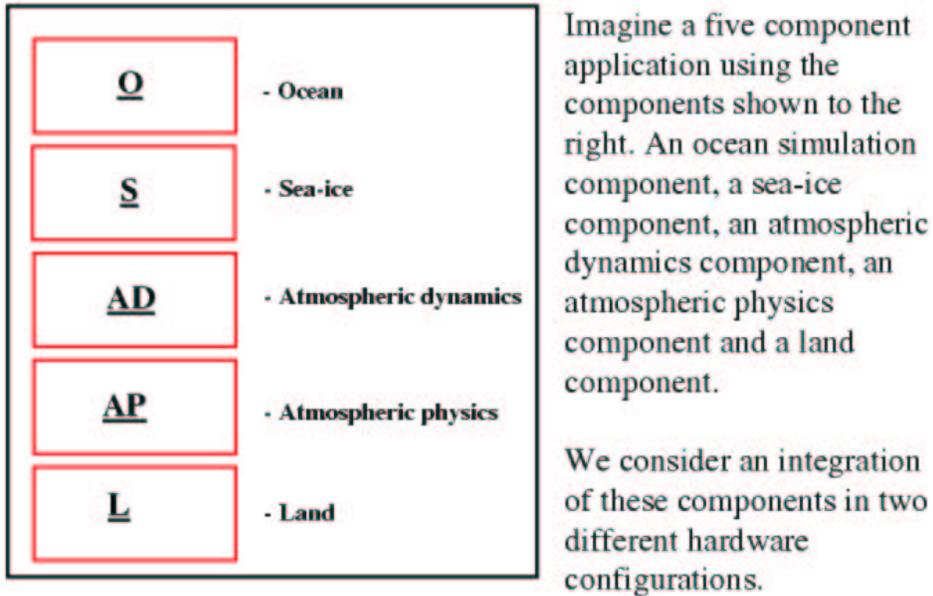


Figure 2:

3.4 Examples

To make the role of Control more concrete we consider some examples of the different configurations that might be assembled into an ESMF Application. The assembly of an application involves both introducing connections between components and deploying components onto hardware. The Control layer in ESMF maintains global information related to the assembly of an applicaton.

The examples shown here illustrate two possible situations that the Control layer has to support.

Consider mapping the application to a single machine with twelve processors, and a hierarchical distributed shared memory configuration. A table specifying this hardware setup is shown below along with a pictorial depiction.

Hardware Resources

Hardware building blocks

$M_1((P_1, P_2, P_3, P_4), (P_5, P_6, P_7, P_8), (P_9, P_{10}, P_{11}, P_{12}))$

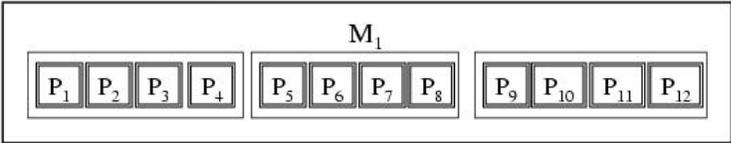


Figure 3:

For this configuration we wish to integrate each component forward sequentially with every component spread over all twelve processors. In our sequence the ocean runs first, then the sea-ice, then the atmospheric dynamics, then the atmospheric physics and finally the land component. This is shown on the next panel

Figure 4:

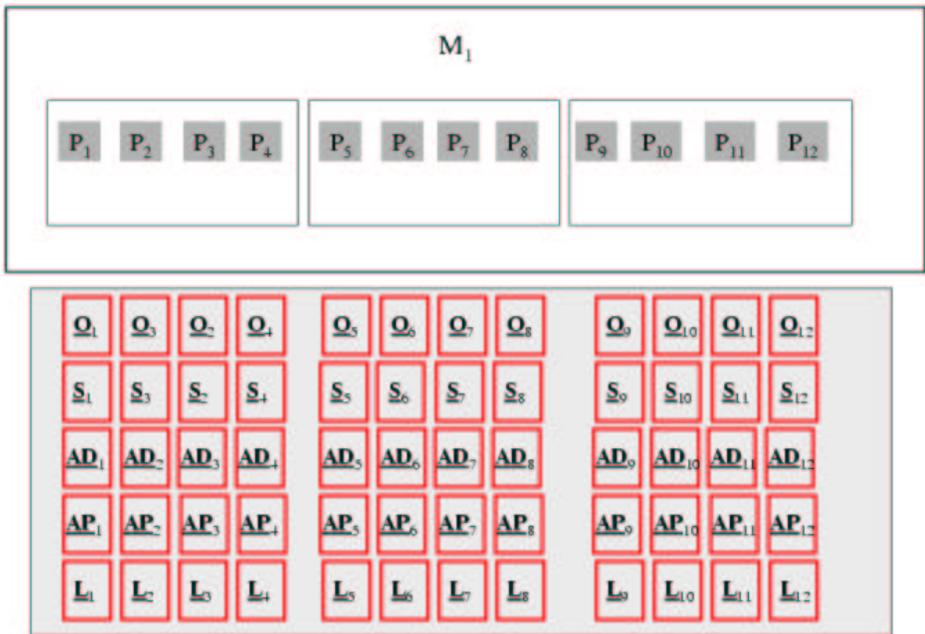


Figure 5:

Consider mapping the application to a configuration that has six machines (M_1 - M_6), two of which have a special interconnect (N_1) and all of which are interconnected by a second network (N_2). Machines M_1 - M_6 have different numbers of processors (P). The “Hardware Resources” table below summarizes the hardware picture at the bottom of the panel.

Hardware Resources

Hardware building blocks

$M_2(P_1) \mid M_3(P_1, P_2) \mid M_4((P_1, P_2), (P_3, P_4)) \mid M_5(P_1, P_2) \mid M_6(P_1, P_2) \mid N_1() \mid N_2()$

Connectivity

$M_1 \ N_1 \ M_2, M_1 \ N_2 \ M_3, M_1 \ N_2 \ M_4, M_1 \ N_2 \ M_5, M_1 \ N_2 \ M_6, M_2 \ N_2 \ M_3, M_2 \ N_2 \ M_4,$
 $M_2 \ N_2 \ M_5, M_2 \ N_2 \ M_6, M_3 \ N_2 \ M_4, M_3 \ N_2 \ M_5, M_3 \ N_2 \ M_6, M_4 \ N_2 \ M_5, M_4 \ N_2 \ M_6,$
 $M_5 \ N_2 \ M_6$

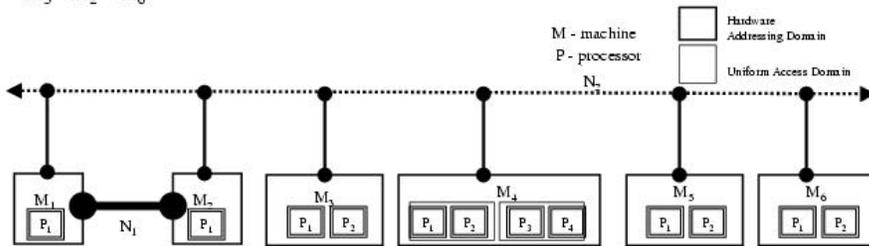


Figure 6:

For this configuration we wish to integrate some components forward concurrently with one another and some sequentially. The ocean and sea ice components integrate forward in sequence on machines M_1 and M_2 . Concurrently, the atmospheric dynamics and then atmospheric physics are integrated forward on machine M_3 and the land component is integrated forward on machines M_4 - M_6 . This is shown on the next panel.

Figure 7:

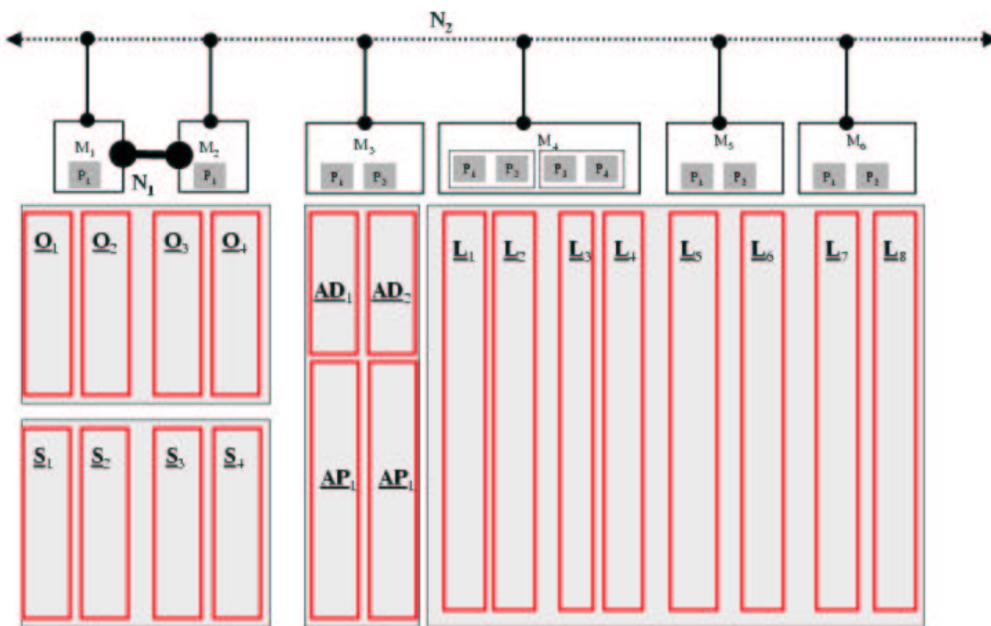


Figure 8:

4 Control requirements

This part covers all the functions for high-level coordination between components.

CTL1 The control element

CTL1.1 Control as main program

The Control element may constitute the main program invoked by each executable participating in an ESMF application.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1

Verification: System test. **Notes:** overlap with Components.

CTL1.2 Control as subroutine

The Control element may be invoked by executables participating in an ESMF application, which have their own main program.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1

Verification: System test. **Notes:** overlap with Components.

CTL1.3 SPMD

It shall be possible to start, advance and terminate all components in the context of a single program.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1

Verification: System test. **Notes:** overlap with Components.

CTL1.4 MPMD

It shall be possible for independently started components to interact with a single control element.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1

Verification: System test. **Notes:** overlap with Components.

CTL1.5 Parallelism

The CNTL element should be scalable.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS (desired), MIT, GFDL

Status: Approved-1.

Verification: System test.

Notes: This is essential for scalability. The control mechanism should not be a sequential bottleneck. **Notes:** overlap with Components.

CTL1.6 Components within address space

The control element should be able to interact with components that share its address space.

Priority: 1

Source: CCSM-CPL, POP, CICE, MIT, GFDL

Status: Approved-1.

Verification: System test. **Notes:** overlap with Components.

CTL1.7 Components outside address space

The control element should be able to interact with components that do not share its address space.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: System test.

Notes: This could be done using MPI based messaging or native API's that are optimal for a given platform. It should be possible for the control layer to make smart decisions about what to use. **Notes:** overlap with Components.

CTL1.8 Customized bindings

The control layer should be able to control the binding of both its own tasks and components' tasks to compute resources, so that the resulting sets of tasks are aligned efficiently on compute resources.

Priority: 2

Source: CCSM-CPL (desired), MIT, GFDL, NSIPP

Status: Approved-1.

Verification: System test.

Notes: We want to have the control layer task for a particular spatial grid region aligned with the physical model components and coupler components that operate on that region. Binding encompasses both associating tasks with specific PE lists and creating the necessary communication substrate to connect tasks that need to communicate. This involves interactions with distributed grid and communication kernel elements. **Notes:** overlap with Components.

CTL1.9 Automatic binding of tasks to resources

Support for automated compute resource binding of tasks should be provided.

Priority: 3

Source: CCSM-CPL (desired), MIT (desired), NSIPP, GFDL

Status: Proposed.

Verification: System test.

Notes: Automated binding could be based on estimates and follow certain optimality rules that are based of both knowledge of compute resources and of the distribution of grids over tasks within multiple, interacting components. Alternatively automated binding could be based on runtime measurement. **Notes:** overlap with Components.

CTL1.10 Write and restore from restart state

The control element shall be able to write its restart state, and be able to reconstruct itself identically based on this state.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: System test. **Notes:** Components must be able to write their own restart. **Notes:** overlap with Components.

CTL1.11 Restore and reconfigure

The control element should be able to restart on a PE configuration different from that in its restart state.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: System test.

Notes: Components must be able to do this too for it to work. **Notes:** overlap with Components.

CTL2 Computational resource management

CTL2.1 Global PE list

It shall be possible, at any instant of an execution, to retrieve the global list of PEs participating in an application.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification:

Notes: Does this grow as MPMD executables check in? Can control retain control of these PEs if MPMD executables check out? **Notes:** overlap with Components.

CTL2.2 Query PE attributes

CTL2.2.1 ID for PEs

Each PE within an application shall have a retrievable unique identifier.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: Unit test. **Notes:** overlap with CommKern.

CTL2.2.2 Memory node ID

Each block of PEs with equal flat access to a block of memory has a retrievable ID shared by those PEs.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: Unit test. **Notes:** overlap with CommKern.

CTL2.2.3 Addressable node ID

A set of PEs capable of addressing the same block of physical memory (e.g., via message passing) has a retrievable ID shared by those PEs.

Priority: 3

Source: CCSM-CPL (desired), GFDL(desired)

Status: Approved-2.

Verification: Unit test. **Notes:** overlap with CommKern.

CTL2.2.4 Processing node ID

A set of PEs to which an operating system scheduler is capable of assigning a single job has a retrievable ID shared by those PEs.

Priority: 3

Source: CCSM-CPL (desired), GFDL(desired)

Status: Approved-2.

Verification: Unit test. **Notes:** Where the OS does not permit this identification, default to PE ID. **Notes:** overlap with CommKern.

CTL2.2.5 Other machine attributes

System communication and I/O profiles will be retrievable.

Priority: 3

Source: CCSM-CPL (desired), MIT(desired)

Status: Approved-2.

Verification: Unit test. **Notes:** overlap with CommKern.

CTL2.3 Creating PE lists

CTL2.3.1 Exact specification

It shall be possible to create a PE list based on a user-specified set of PE IDs.

Priority: 1

Source: CCSM-CPL, PSAS, MIT, GFDL

Status: Approved-1.

Verification: Unit test.

Notes: For example, "Component requests PEs 0...31." **Notes:** overlap with Components.

CTL2.3.2 Length specification

It shall be possible to create a PE list of a user-specified length (PE count).

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: Unit test.

Notes: For example, "Component requests N PEs." Exception if N too large. **Notes:** overlap with Components.

CTL2.3.3 “Free” specification

It is possible to create a PE list of a user-specified length on which no other components have been scheduled.

Priority: 3

Source: CCSM-CPL (desired), PSAS (desired), MIT(desired), GFDL(desired)

Status: Approved-2.

Verification: Unit test.

Notes: For example, “Component requests N free PEs.” Exception if N too large. **Notes:** overlap with Components.

CTL2.3.4 Memory node affinity

It shall be possible to create PE lists based on memory node affinity.

Priority: 3

Source: CCSM-CPL (desired), MIT(desired), GFDL(desired)

Status: Approved-2.

Verification: Unit test.

Notes: For example, “component requests N PEs sharing an memory node”. Throw exception if length is greater than number of PEs on the memory node. Other affinities may be required e.g. scratch space, memory, same CPU, particular numerical library etc..... **Notes:** overlap with Components.

CTL2.3.5 Addressable node affinity

It is possible to create PE lists based on addressable node affinity.

Priority: 3

Source: CCSM-CPL (desired), MIT(desired), GFDL(desired)

Status: Approved-2.

Verification: Unit test.

Notes: For example, “component requests N PEs sharing an addressable node”. Throw exception if length is greater than number of PEs on the addressable node.

CTL2.3.6 Processing node affinity

It is possible to create PE lists based on processing node affinity.

Priority: 3

Source: CCSM-CPL (desired), MIT(desired), GFDL(desired)

Status: Approved-2.

Verification: Unit test.

Notes: For example, “component requests N PEs sharing a processing node”. Throw exception if length is greater than number of PEs on the processing node. **Notes:** overlap with Components.

CTL2.3.7 Component affinity

It is possible to create PE lists based on component affinity. This is to assign components which are closely linked to be assigned “nearby” computational elements.

Priority: 2

Source: CCSM-CPL (desired), POP (desired), CICE (desired), PSAS (desired), MIT(important - it could be hard to meet the 10% degradation milestone without this control somewhere), GFDL **Status:** Approved-2.

Verification: Unit test.

Notes: For example, “atmospheric physics component would like to be scheduled next to atmospheric dynamics”. **Notes:** overlap with Components.

CTL2.3.8 Compact PE list

It is possible to request a PE list of given length whose associated memory extent is optimally compact.

Priority: 3

Source: CCSM-CPL (desired), MIT(desired), GFDL(desired)

Status: Approved-2.

Verification: Unit test?

Notes: For example, “give me N PEs distributed on the fewest possible memory nodes”. Other attributes could be optimized in deciding layout. A general graph and edge optimizer could be used. **Notes:** overlap with Components.

CTL2.3.9 Spawning of PE lists

It is possible to request a PE list subset of an existing PE list.

Priority: 2

Source: CCSM-CPL (desired), PSAS (desired), MIT, GFDL

Status: Approved-2.

Verification: Unit test.

Notes: “Of the 80p assigned to ocean+atmosphere, I’d like 32 for the atmosphere”. **Notes:** overlap with Components.

CTL2.3.10 Complementary PE lists

It is possible to request a complementary subset of an existing spawned PE list with respect to its parent.

Priority: 2

Source: CCSM-CPL (desired), MIT, GFDL

Status: Approved-2.

Verification: Unit test.

Notes: “...and the other 48 for the ocean”. Required for concurrent scheduling. **Notes:** overlap with Components.

CTL2.3.11 Modify PE list request

It is possible for a component to make any of the above requests at any time.

Priority: 3

Source: CCSM-CPL (desired), MIT(desired), GFDL(desired)

Status: Proposed.

Verification: System test. **Notes:** overlap with Components.

CTL3 Memory management

This overlaps somewhat with requirements in the **CommKernel** layer. I am including one line item here:

CTL3.1 Mapped memory region for data sharing

It shall be possible to request a region of shared address space to support cross-component communication.

Priority: 1

Source: MIT(needed for 10% milestone), GFDL

Status: Approved-1.

Verification: Unit test.

Notes: This is somewhat obscure: it is possible within shared address space to declare a region of memory that all participating PEs see the same way (i.e pointers can be shared). This allows data sharing between components to happen with much lower latencies than message-passing, albeit with some requirement on the framework to accommodate different platform memory consistency behavior. This requirement is to lay the ground to support such semantics. Its also possible between processing nodes(e.g. TreadMarks, SCI, VIA, RDMA, IMC) or memory nodes (of course)!!! **Notes:** overlap with Components.

CTL4 Component initiation

How components get scheduled.

CTL4.1 Instantiation

It must be possible to create any number of instances of a component.

Priority: 1

Source: CCSM-CPL (desired), MIT(important), GFDL(important)

Status: Approved-1.

Verification: System test.

Note: Max/Michelle is this needed for NSIPP EnKF - CNH? **Notes:** overlap with Components.

CTL5 Component termination

What happens when a component is done.

CTL5.1 Termination

It must be possible to destroy any instance of a component.

Priority: 1

Source: CCSM-CPL (desired), MIT, GFDL

Status: Approved-1.

Verification: System test. **Notes:** overlap with Components.

CTL5.2 Release PE list

It shall be possible for a component to release its pelist.

Priority: 1

Source: CCSM-CPL (desired), MIT, GFDL

Status: Approved-1.

Verification: System test. **Notes:** overlap with Components.

CTL5.3 Component disappearance

Whenever possible, the control element should attempt a graceful exit (e.g checkpointing of all “live” components) when a component under control undergoes unscheduled termination.

Priority: 1

Source: CCSM-CPL (desired), POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: System test.

CTL6 Component scheduling

This section covers ways in which components may be directed to advance.

CTL6.1 By timestep

It shall be possible to direct components to advance a specific number of steps.

Priority: 2

Source: CCSM-CPL, PSAS, MIT, GFDL(some codes)

Status: Approved-2.

Verification: System test.

Notes: Not all components may support this option. **Notes:** overlap with Components.

CTL6.2 By time

It shall be possible to direct components to return to control at a specified time instant.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1

Verification: System test.

Notes: Not all components may support this option. **Notes:** overlap with Components.

CTL6.3 By alarm

It shall be possible to direct components to return to control at a specified alarm event.

Priority: 2

Source: CCSM-CPL, POP, CICE, PSAS, MIT

Status: Approved-2.

Verification: System test. **Notes:** overlap with Components.

CTL7 Cross-component scheduling

Scheduling relationships between multiple components.

CTL7.1 Exchange scheduling

It is possible for two components to schedule data exchanges at regular intervals.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: System test.

Notes: Regularity is probably not a requirement: any semantics supported by **TM:Alarm** should be possible.

Notes: overlap with Components.

CTL7.2 Serial execution

It shall be possible to schedule components for serial execution.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: System test. **Notes:** overlap with Components.

CTL7.3 Concurrent execution

It shall be possible to schedule components for concurrent execution.

Priority: 1

Source: CCSM-CPL, POP, CICE, PSAS, MIT, GFDL

Status: Approved-1.

Verification: **Notes:** overlap with Components.

CTL7.4 Forward coupling timestep

The coupling timestep between two components may be *forward* in time.

Priority: 2

Source: GFDL, MIT

Status: Approved-2.

Verification: System test.

Notes: i.e., given two components C_1 and C_2 ,

$$C_2^{t+1} = C_2^t + \text{other terms} + f(C_1^t) \quad (1)$$

$$C_1^{t+1} = C_1^t + \text{other terms} + f(C_2^t) \quad (2)$$

Notes: compress into notes.

CTL7.5 Backward coupling timestep

The coupling timestep between two components may be *backward* in time.

Priority: 2

Source: GFDL, MIT

Status: Approved-2.

Verification: System test.

Notes: i.e., given two components C_1 and C_2 ,

$$C_2^{t+1} = C_2^t + \text{other terms} + f(C_1^{t+1}) \quad (3)$$

$$C_1^{t+1} = C_1^t + \text{other terms} + f(C_2^{t+1}) \quad (4)$$

Notes: compress into notes.

CTL7.6 Forward-backward

The coupling timestep between two components may be *forward-backward* in time.

Priority: 2

Source: GFDL, MIT

Status: Approved-2.

Verification: System test.

Notes: i.e., given two components C_1 and C_2 ,

$$C_2^{t+1} = C_2^t + \text{other terms} + f(C_1^t) \quad (5)$$

$$C_1^{t+1} = C_1^t + \text{other terms} + f(C_2^{t+1}) \quad (6)$$

Notes: compress into notes.

CTL8 Cross-component exchange

CTL8.1 Explicit exchange

It shall be possible to request an explicit data exchange.

Priority: 3

Source: PSAS (desired), MIT, GFDL

Status: Proposed.

Verification: System test.

Notes: When the scheduling timestep in CTL7 is consistent with explicit exchange. **Notes:** compress into notes.

CTL8.2 Implicit exchange

It shall be possible to request an implicit data exchange.

Priority: 3

Source: GFDL

Status: Proposed.

Verification: System test.

Notes: When the scheduling timestep in CTL7 is consistent with implicit exchange. What is one of these - CNH? That is, communication would include the implicit sensitivity terms that occur, for example, in a tridiagonal solver. **Notes:** compress into notes.

CTL9 Cross-component signals

Signalling is used for events that may occur within a component at unscheduled, or unpredictable intervals. Scheduled events have already been dealt with in the cross-component requirements of **TM:Alarm**. Components should be able to issue and trap signals.

CTL9.1 Checkpoint request

It shall be possible to send a signal to a component asking it to halt.

Priority: 1

Source: CCSM-CPL, PSAS, MIT, GFDL

Status: Approved-1.

Verification: System test. **Notes:** overlap with Components.

CTL9.2 Data request

It shall be possible to send a signal to a component asking for data at a specified time instant.

Priority: 2

Source: CCSM-CPL, PSAS, MIT, GFDL

Status: Approved-2.

Verification: System test. **Notes:** overlap with Components.

CTL10 Syntax

CTL10.1 Common syntax across platforms

The control syntax should be the same on all platforms.

Priority: 1

Source: CCSM-CPL, PSAS, MIT, GFDL

Status: Approved-1.

Verification: Code inspection.

CTL10.2 Performance

The control mechanism should be compatible with 100msec component cycling times.

Priority: 2

Source: MIT

Status: Approved-2.

Verification: System test.

Part III

Superstructure: Components

1 Authors, target codes and review team

Authors: Max Suarez, Cecelia DeLuca, Jay Larson

Review Date: 16 May, 2002

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CCSM-CPL	Kauffman, Jones
CAM-FV, PSAS	da Silva, Sawyer
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NSIPP-ODA	Keppenne
NCEP-ATM, SSI	Iredell, Young

Other Reviewers: DeLuca, Neckels, Larson, Jacob

2 Component background

An important function of ESMF is to support the integration of complex computational modules developed by earth scientists into larger models and data assimilation systems, and to facilitate the interoperability of these modules. In the ESMF design, these integration functions are achieved by combining Couplers and Gridded Components into Integrated Applications.

Gridded Components will contain the computational modules and will thus be the main arena for interaction between computational scientists and the Framework. Couplers will provide all necessary functionality required to integrate Gridded Components into a larger application (e.g., ocean and atmosphere components into a climate model).

Following the ESMF general design, Couplers and Gridded Components are to be implemented using an object-oriented approach in which the data and actions of these components are encapsulated in ESMF specified ways.

2.1 Location

Couplers and Gridded Components, being key elements in the Framework's integration functions fall in the Superstructure. They will be invoked in ESMF specified ways by Integrated Applications and will, in turn, rely on other Superstructure and Infrastructure elements in their internal design. Coupling between computational modules, for example, often requires interpolation of data and its reorganization in memory. To perform these tasks, couplers will rely on Superstructure Regridding methods, which in turn rely on the Infrastructure's uniform implementation of various machine memory organizations and user-chosen data decompositions.

2.2 Scope

Requirements on gridded components represent a compromise between the desire to facilitate their interoperability and the need for computational modules to have essentially no restriction on their physical content or computational approach. A fully interoperable framework would have to prescribe the actual physical quantities to be exchanged by components, as well as placing strong inclusive and exclusive restrictions on the physics that they represent. The

General Requirements preclude ESMF from achieving this “strong” interoperability. ESMF is intended for diverse applications and must support the rapidly evolving models and data handling techniques. The scope of requirements on components will thus be limited to the way in which data is exchanged between them and some restriction on how their internal data must be structured to access Framework services and to allow its instantiation by Integrated Applications. They will *not* include requirements on what physical quantities components must provide or accept, or what calculations they must perform.

CGC1 Component requirements

CGC2 Components

CGC2.1 Component names

Each component shall be associated with a name. If a name is not supplied by the user at creation, one will be assigned by default. User-defined component names shall not conflict with names reserved for ESMF generic components.

Priority: 1

Source: Required by NSIPP, MIT, CCSM-CPL

Status: Approved-1.

Verification: Unit test.

CGC2.2 Creation

CGC2.2.1 Standard creation

It shall be possible to create a component based on a PE list or layout, and a set of other arguments that are optional or component specific.

Priority: 1

Source: Required by NSIPP, MIT, CCSM-CPL, GFDL

Status: Approved-1.

Verification: Unit testing.

CGC2.2.2 Creation based on replication

It shall be possible to create a component by specifying an existing component to be replicated, an optional name, and an optional PE list or layout. If no PE list or layout is supplied one from the component being replicated shall be used.

Priority: 2

Source: Required by NSIPP, MIT, GFDL(desired)

Status: Approved-2.

Verification: Unit testing.

Notes: A typical use would be to perform an ensemble of integrations within a single application. This is required by applications that have tight coupling between ensemble members, such as ensemble Kalman Filter.

CGC2.2.3 Creation of subcomponents

It shall be possible for a component to create a subcomponent using the component creation methods above. The subcomponent must be defined on a PE list or layout that is entirely contained within that of the component that creates it.

Priority: 2

Source: Required by MIT, GFDL

Status: Approved-2.

Verification: Unit test.

CGC2.3 Interaction with subcomponents

A component shall be able to call any public method of its subcomponents. A component is responsible for coordinating the execution of its subcomponents, for example by ensuring that they are properly initialized, run and halted.

Priority: 2
Source: MIT, GFDL
Status: Approved-2.
Verification: Code inspection.

CGC2.4 Deletion

A component may be deleted, in which case it is expected to delete its subcomponents before deleting itself. The component being deleted shall free any memory that it has allocated.

Priority: 2
Source: Required by NSIPP, MIT, GFDL.
Status: Approved-2.
Verification: Unit test.

CGC2.5 Operations

CGC2.5.1 Initialize

A component shall have a method that initializes its internal values.

Priority: 1
Source: Required by NSIPP, MIT, CCSM-CPL, GFDL.
Status: Approved-1.
Verification: System test.

CGC2.5.2 Run

A component shall have a run method that initiates its execution and allows the user to specify the running interval.

Priority: 1
Source: Required by NSIPP, MIT, CCSM-CPL, GFDL.
Status: Approved-1.
Verification: System test.

CGC2.5.3 Halt

A component shall have a halt method that saves its restart state and gracefully terminates its execution.

Priority: 1
Source: MIT, GFDL
Status: Approved-1.
Verification: System test.

CGC2.5.4 Prepare output exchange packets

A component shall provide a method that prepares an output exchange packet based on selections from its export state.

Priority: 1
Source: Required by MIT, CCSM-CPL, NSIPP, GFDL.
Status: Approved-2.
Verification: System test.

CGC2.5.5 Accept input exchange packets

A component shall provide a method that accepts an input exchange packet in conformance with its import state.

Priority: 1
Source: MIT, GFDL
Status: Approved-2.
Verification: System test.

CGC2.5.6 Write and restore from restart

A component shall be able to write its restart state and be able to reconstruct itself identically based on that state.

Priority: 1
Source: MIT, GFDL
Status: Approved-1.
Verification: System test.

CGC2.6 Queries

CGC2.6.1 Query name

A component shall be able to return its name.

Priority: 3
Source: MIT
Status: Proposed.
Verification:

CGC2.6.2 Query layout

A component shall be able to return the layout over which it is distributed, including its PE list and its communication and data sharing mechanisms. Whether or not the following are in use and how they are each bound to PEs may be queried: MPI processes, concurrent Unix processes, concurrent OpenMP threads, concurrent Posix threads.

Priority: 2
Source: CCSM-CPL, PSAS, MIT, GFDL
Status: Approved-2.
Verification:

CGC2.6.3 Query run status

Components shall provide a method that reports whether the component is initialized, running, halted, or in an error condition.

Priority: 1
Source: NSIPP, MIT, GFDL
Status: Approved-1.
Verification:

CGC2.6.4 Query subcomponent layout

A component shall be able to return the layouts of its subcomponents.

Priority: 2
Source: MIT, GFDL
Status: Approved-2.
Verification:

CGC2.6.5 Query import state

A component shall be able to return a list and description of the data, such as field names and descriptions, that it requires as input in order to run. This shall be provided in a standard format.

Priority: 1
Source: NSIPP, MIT, CCSM-CPL, POP, CICE, GFDL
Status: Approved-2.
Verification:

CGC2.6.6 Query export state

A component shall be able to return a list and description of the data that it can make available to other components. This shall be provided in a standard format.

Priority: 1
Source: NSIPP, MIT, CCSM-CPL, POP, CICE, GFDL
Status: Approved-2.
Verification:

CGC2.6.7 Query state summary

Each component shall provide a query method that returns its run status (e.g. whether the component has been initialized), the size of its restart state, if initialized, appropriate verification checksums, and any control parameters that the component deems “application-settable.”

Priority: 2
Source: NSIPP
Status: Approved-2.
Verification:

CGC2.7 Query exchange packets

Given the name of an exchange packet, a component shall return a reference to that exchange packet. A component may have multiple exchange packets defined.

Priority: 3
Source: Desired by MIT
Status: Proposed.
Verification:

CGC2.7.1 Query input datasets

A component shall be able to return a list of any input data files necessary for it to run. If this information is not available it shall relay that information to the user.

Priority: 3
Source: MIT
Status: Proposed.
Verification:

CGC2.7.2 Query compute parameters

Insofar as it is possible, a component shall be able to return a description of the computing environment in which it is being run, including computing architecture and compiler name and version, compile options, and numerical precision.

Priority: 3
Source: MIT
Status: Proposed.
Verification:

Notes: This requirement may be extended and further detailed in the parameter specification requirements document.

CGC2.7.3 Consolidated query responses

A component shall be able to direct queries to its subcomponents and return consolidated replies. For example, a component shall be able to return a list of the names of all the subcomponents that it contains, a list of subcomponent layouts, and an overall run status based on the individual run statuses of its subcomponents.

Priority: 3
Source: MIT, GFDL
Status: Approved-2.
Verification:

CGC3 Application components

In addition to the general requirements for a component described in CGC2, application components are required to provide the following capabilities.

CGC3.1 Application initialize operation

An application shall provide a method that allocates and initializes resources, such as underlying libraries and any global variables or buffer space, necessary to run an ESMF application.

Priority: 1
Source: MIT, DAO, GFDL
Status: Approved-1.
Verification:

CGC3.2 Queries

CGC3.2.1 Query case name

An application shall return its case name, the user-specified name for a particular run. If no case name is specified a default will be assigned.

Priority: 3
Source: MIT, DAO
Status: Proposed.
Verification:

CGC3.2.2 Query case date

An application shall return the real-world date at which its execution began.

Priority: 3
Source: MIT
Status: Proposed.
Verification:

CGC4 Gridded components

In addition to the general requirements for a component described in CGC2, gridded components are required to provide the following capabilities.

CGC4.1 Gridded components have one or more associated grids

A gridded component shall be associated with one or more grids. A default grid shall be associated with each gridded component and shall default to the first grid created in the component. The default grid may be assigned.

Priority: 1
Source: NSIPP, GFDL, MIT
Status: Approved-1.
Verification:

CGC4.2 Creation

The creation method of a gridded component shall require as an argument either a fully created grid or the parameters required to create a grid.

Priority: 1
Source: Required by NSIPP, MIT, CCSM-CPL, GFDL
Status: Approved-1.
Verification: Unit test.

CGC4.3 Queries

CGC4.3.1 Query grids

A gridded component can return a list of its grids.

Priority: 3
Source:
Status: Proposed. **Verification:** Unit test.

CGC4.3.2 Query default grid

A gridded component can return its default grid.

Priority: 2

Source: GFDL, MIT

Status: Approved-2.

Verification: Unit test.

CGC5 Coupler components

In addition to the general requirements for a component described in CGC2, coupler components are required to provide the following capabilities.

CGC5.1 Coupler run operation

A coupler component shall provide a run method in which the data in an output exchange packet of a source component is transformed and transferred to a destination component where it again is transformed into an input exchange packet. This operation may occur at a given event or a set interval. Transforms may be user-defined, generic, or null, and multiple transforms may be applied in succession.

Priority: 1

Source: NSIPP, MIT, GFDL

Status: Approved-1.

Verification: System test.

CGC5.1.1 Coupling operation limited to two components

A single coupling operation shall be limited to data transformation and transfer between two components.

Priority:

Source: NSIPP

Status: Rejected.

Verification:

Notes: We need to define coupling between more than two components, for example when broadcasting values to ensembles, generating ensemble means, or merging the spans of multiple components to interface with another whose span is their union (e.g. combining ocean and land to form a complete interface with the atmosphere).

CGC5.1.2 Unlimited number of coupling operations

No limit is placed on the number of coupling operations defined or executed between a pair of component or in an application overall, other than computing resources.

Priority: 1

Source: MIT, GFDL

Status: Approved-1.

Verification: Unit test.

CGC6 General computational requirements

CGC6.1 Validity checking

Methods shall be provided to check the validity of components and coupling operations and report inconsistencies or errors.

Priority: 2

Source: NSIPP

Status: Approved-2.

Verification: System test.

CGC6.2 Compute overhead

The component interface mechanisms need to be compatible with operation frequency as high as ten times per second (i.e. the period during which no component is in an interface phase may be as short as 100msec).

Priority: 3

Source: MIT

Status: Proposed.

Verification: System test.

Part IV

Infrastructure Fields and Grids: Fields

1 Authors, target codes and review team

Authors: Cecelia DeLuca, Tony Craig, Chris Hill

Review Date: 23 April, 2002

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CAM-EUL, CLM	Craig
CAM-FV, PSAS	da Silva
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell, Young

Other Reviewers: DeLuca, Neckels, Larson, Jacob

2 Background

The capabilities most fundamental to ESMF are the efficient transformation and transferral of field data between model components. Fields within a model component are frequently associated with the same physical grid (though staggering may be different) and similarly decomposed in memory. Thus the basic currency exchanged between components is a bundle of fields associated with the same **grid**, or simply a **bundle**.

2.1 Location

Fields and bundles are part of the Fields and Grids layer of the ESMF Infrastructure.

2.2 Scope

Fields may be defined in physical or spectral space, and may be scalar or vector.

Field and bundle operations are limited to configuration, manipulation and querying of attributes, accessing data, setting and retrieving data values, I/O, and regriding. The first releases of ESMF will not provide differential or other mathematical operators on fields; the framework may be extended to include such methods in the future.

3 Field summary of requirements

4 Field requirements

FLD1 Fields

FLD1.1 Creation

FLD1.1.1 Creation with data allocation

Fields may be created by specifying attributes, a grid, data array dimensions and descriptors, optional masks (e.g. for active cells), and an optional I/O specification. In this case a field will allocate its own data. The grid passed into the argument list is referenced and not copied.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.1.2 Creation with external data

Fields may be created as in FLD1.1.1 with a data array passed into the argument list. The data array is referenced and not copied.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.1.3 Creation without data

Fields may be created as in FLD1.1.1 without allocating data or specifying an associated data array. In this case specifying the grid parameters and data array dimensions may be deferred until data is attached.

Priority: 1

Source: CCSM-CPL, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.1.4 Creation by indexing an existing field

Fields may be created by specifying some subset of the domain of an existing field. The user may specify whether the original field's data is copied or referenced. The grid is referenced. Staggering and mask of active/inactive cells may be different in the original and new fields.

Priority: 2

Source: CCSM-CPL, CAM-FV, PSAS, MIT, WRF.

Status: Approved-2.

Verification: Unit test.

FLD1.1.5 Creation with remap

Fields may be created as a result of transposing, interpolating, or regridding an existing field. The original field's data is copied, but the target grid must already exist.

Priority: 2

Source: CCSM-CPL, MIT, WRF, GFDL.

Status: Approved-2.

Verification: Unit test.

Notes: The assumption here is that the grid is being altered, so it should be copied and not referenced. Implies adjoint form.

FLD1.1.6 Creation by weighted combination

Fields may be created as the result of weighting (e.g. temporally or spatially) combinations of existing field values. The original fields and the resulting field are defined on the same grid. The original field's grid is referenced. Implies adjoint form - maybe always self-adjoint (not sure) - CNH.

Priority: 2

Source: CCSM-CPL, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

FLD1.2 Local memory layout

It shall be possible to specify the local memory layout (major axis) of field data at field creation and to rearrange it (assumes local copy).

Priority: 1

Source: CCSM-CPL, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.3 Deletion

Fields may be deleted.

Priority: 1

Source: CCSM-CPL, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: This implies a reference count will be defined for grids.

FLD1.4 Attributes

The user can define a list of attribute name and value pairs, such as `units`, `meters`, for a field. The attributes may be of character, real, integer, or logical type, or arrays of any of these.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.4.1 Default attributes

The only default attribute of a field will be a name.

Priority:

Source: CCSM-CPL, CAM-FV, PSAS, MIT, WRF.

Status: Rejected.

Verification: Unit test. **Notes:** For consistency with the IO requirements more than name may be mandatory (Arlindo).

FLD1.4.2 Recommended attributes

ESMF shall provide a list of recommended attribute names and values.

Priority: 2

Source: CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-2.

Verification: Unit test.

Notes: This list likely to be based on the CF convention. For consistency with IO these may be mandatory, not required (Arlindo). (What does "mandatory, not required" mean - CNH).

FLD1.4.3 Add and delete attributes

Attributes may be added to or deleted from a given field. The name attribute cannot be deleted.

Priority: 2

Source: CCSM-CPL, POP, CICE, MIT, WRF, GFDL.

Status: Approved-2.

Verification: Unit test.

FLD1.4.4 Copy attributes

Attributes may be copied from one field to another.

Priority: 2

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT(desired), GFDL.

Status: Approved-2.

Verification: Unit test.

FLD1.4.5 Collective assignment of attributes

Attributes may be added to or deleted from a list of fields collectively.

Priority: 2

Source: CCSM-CPL, WRF.

Status: Approved-2.

Verification: Unit test.

Notes: The list of fields for which attributes are collectively defined does not need to be in the same bundle.

FLD1.5 Operations

FLD1.5.1 Remap data

It shall be possible to regrid, interpolate, or redistribute field data subject to the requirements for those operations.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.5.2 Return grid

A field shall be able to return a reference to its grid.

Priority: 1

Source: CCSM-CPL, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: It is assumed that a grid contains both a distributed grid and a physical grid.

FLD1.5.3 Return local memory layout

A field shall be able to return a description of its local memory layout.

Priority: 1

Source: CCSM-CPL, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.5.4 Direct data access

Data arrays may be easily detached or attached to a field. When data is detached from a field it is not deallocated; the user receives a reference to the data array. The field shall be able to identify whether a data segment is attached or detached. The types of access supported shall be contiguous whole array or array subset and strided whole array or array subset.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: The expectation here is that if a user wants to directly access field data it must be detached from the field first. (Unless they want to dereference the field, which is their own business...)

FLD1.5.5 Data access via copy

It shall be possible to retrieve an array that is a copy of all or a subset of the data values associated with a field.

Priority: 1

Source: CCSM-CPL, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.5.6 Set

Field data may be set by specifying an index or coordinate range and a data value.

Priority: 1

Source: CCSM-CPL, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes This data access is not direct, as with attach and detach - it is through the field interface. It enables setting single data elements and arrays to a given single value, for example, for initialization.

FLD1.5.7 Write and restore from restart

Methods shall be provided that enable a field to write out restart data and to reconstruct itself identically from that data.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.6 Queries

FLD1.6.1 Query name

A field shall be able to easily return its name. If the user does not provide a field name one will be created. Field names must be unique within an address space and it shall be possible to check this.

Priority: 1

Source: CCSM-CPL, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.6.2 Query number of dimensions

A field shall be able to return the number of dimensions it has.

Priority: 1

Source: CCSM-CPL, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.6.3 Query attributes

A field can return its list of attributes.

Priority: 1

Source: CCSM-CPL, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.6.4 Query attributes by name

A field shall be able to return attribute values given a list of attribute names.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.6.5 Query number of attributes

A field shall be able to return the number of attributes it possesses.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.6.6 Query presence of data

It shall be possible to determine whether a field has associated data.

Priority: 1

Source: CCSM-CPL, POP, CICE, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD1.6.7 Query number of local/global cells or gridpoints

A field may be queried for the number of local or global cells or gridpoints in its underlying grid.

Priority: 1

Source: CCSM-CPL, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD2 Bundles

FLD2.1 Creation

FLD2.1.1 Creation using field list

It shall be possible to create a bundle with a field list, an optional I/O specification, and an identifier that specifies whether the bundle is to be packed (contiguous data) or loose (noncontiguous data). Bundles within

Priority: 1

Source: CCSM-CPL, POP, CICE.

Status: Approved-1.

Verification: Unit test.

FLD2.1.2 Creation by indexing an existing bundle

It shall be possible to initialize a bundle that is a copy of an existing bundle, or a specified subset of the fields or domain of an existing bundle. The grid associated with the bundle shall be referenced and not copied.

Priority: 1

Source: CCSM-CPL.

Status: Approved-1.

Verification: Unit test.

FLD2.1.3 Creation with remap

Bundles may be created as the result of a remapping operation on an existing bundle. The grid in this case is copied.

Priority: 1

Source: CCSM-CPL.

Status: Approved-1. Implies possible adjoint form

Verification: Unit test.

FLD2.2 Local memory layout

It shall be possible to specify the local memory layout (major axis, field interleaving) of data within a packed bundle.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.

FLD2.3 Deletion

Bundles may be deleted. Fields within a bundle will be deleted if they are not being used elsewhere.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.

FLD2.4 Operations

FLD2.4.1 Remap data

It shall be possible to regrid, interpolate, or redistribute bundle data subject to the requirements for those operations.

Priority: 1
Source: CCSM-CPL, POP, CICE.
Status: Approved-1.
Verification: Unit test.

FLD2.4.2 Insert and remove field

A field may be inserted into or removed from a bundle.

Priority: 1
Source: CCSM-CPL, POP.
Status: Approved-1.
Verification: Unit test.
Notes: Inserting a field into a bundle may result in a reallocation and copy if the bundle is packed.

FLD2.4.3 Direct data access

Data arrays may be easily detached or attached to a bundle. When data is detached from a bundle it is not deallocated; the user receives a reference to the data array. The bundle shall be able to identify whether a data segment is attached or detached. The type of access supported shall be contiguous whole array or offset by field.

Priority: 1
Source: CCSM-CPL, POP, CICE.
Status: Approved-1.
Verification: Unit test.

FLD2.4.4 Data access via copy

It shall be possible to retrieve an array that is a copy of all or a subset of the data values associated with a bundle.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.

FLD2.4.5 Set

Bundle data may be identified in its entirety or by index or coordinate and set to a single value, for example, for initialization.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.

FLD2.4.6 Return field(s)

A bundle shall be able to return a pointer to a field indexed by name or pointers to all the fields that it contains.

Priority: 1
Source: CCSM-CPL, POP, CICE.
Status: Approved-1.
Verification: Unit test.

FLD2.4.7 Return grid

A bundle shall be able to return a pointer to its grid.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.
Notes: It is assumed that a grid contains both a distributed grid and a physical grid.

FLD2.4.8 Return local memory layout

A bundle shall be able to return a description of its local memory layout.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.

FLD2.4.9 Pack bundle

A bundle shall be able to create a copy of its field data that is locally contiguous.

Priority: 1
Source: CCSM-CPL
Status: Approved-1.
Verification: Unit test.

FLD2.4.10 Write and restore from restart

Methods shall be provided that enable a bundle to write out restart data and to reconstruct itself identically from that data.

Priority: 1
Source: CCSM-CPL, POP, CICE.
Status: Approved-1.
Verification: Unit test.

FLD2.5 Queries

FLD2.5.1 Query bundle name

A bundle shall be able to return its name. If the user does not assign a name one will be assigned by default. All bundle names within an address space must be unique, and it shall be possible to check this.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.

FLD2.5.2 Query field names

A bundle shall be able to return a list of field names.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.

FLD2.5.3 Query number of fields

A bundle shall be able to return the number of fields that it contains.

Priority: 1
Source: CCSM-CPL, POP, CICE.
Status: Approved-1.
Verification: Unit test.

FLD2.5.4 Query number of local/global cells or gridpoints

A bundle may be queried for the number of local or global cells or gridpoints.

Priority: 1
Source: CCSM-CPL.
Status: Approved-1.
Verification: Unit test.

FLD3 Field and bundle I/O

FLD3.1 Write

It shall be possible to write the data values of a field or bundle to a specified destination.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: A single IO capability may be sufficient for this and the field restart requirement (Arlindo).

FLD3.2 Set destination

It shall be possible to easily customize the destination of a field or bundle write operation.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: We should at least lay the foundation for URI type naming etc... (for both read and write).

FLD3.3 Set write frequency

It shall be possible to specify the frequency, using a number of timesteps or a time interval, at which either field or bundle data is written. The default shall be no writes, and it shall be possible for the user to reset the default.

Priority: 1

Source: CCSM-CPL, POP, CICE, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD3.4 Write indexed values

It shall be possible to specify and to easily reset a subset of field and bundle data and attributes to be written out.

Priority: 2

Source: CCSM-CPL, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

FLD3.5 Set precision

It shall be possible to specify the precision at which either field or bundle data is written.

Priority: 1

Source: CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

FLD4 General computational requirements

FLD4.1 Validity checking

It shall be possible to check fields and bundles for internal consistency, and to disable the checks for optimization purposes.

Priority: 2

Source: CCSM-CPL, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

Part V

Infrastructure Fields and Grids: Physical Grids

1 Authors, target codes and review team

Authors: Robert Hallberg, Arlindo da Silva, Matt Harrison

Review Date: 16 April, 2002

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CCSM grids	Craig, Jones
CAM-FV, PSAS	da Silva
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell

Other Reviewers: DeLuca, Neckels, Larson, Jacob

2 Background

Earth system models use a variety of different discrete grids to represent continuous physical space. The ESMF physical grid is responsible for maintaining information about physical coordinate based discretizations of simulation domains. Components may include and use additional grid information internally, however, the only physical grid information that framework operates on will come through physical grid elements.

The information that the framework needs to represent in physical grids is quite extensive. Coupled components need to be able to provide physical grid information that is sufficiently detailed for regridding operators (see Regrid) which transfer field data between the components. The primary ESMF codes employ finite-difference and finite-volume grids, spectral grids, unstructured land-surface grids and ungridded observational networks. This requires the physical grid to support an extensive range of metric terms (e.g. grid spacings, areas and volumes) and grid masks.

The grids used in Earth system models evolve continually. The physical grid of ESMF will be extensible, so that support for future grids can be added to the framework over time.

2.1 Location

The physical grid is a part of the "Fields and Grids" portion of the ESMF infrastructure layer. The physical grid interacts closely with the index spaces in the distributed grid facility; the creation of a distributed grid precedes the creation of a physical grid. Most Fields exist at locations described via physical grids. Regrid typically uses physical grid information. The ESMF Control superstructure element supports the passing of physical grid information between components. Field values with different physical grid settings and belonging to different components can be exchanged through the Coupler element of the superstructure.

2.2 Scope

Physical grids do not cover domain decomposition or specification of grid topology; these are covered by distributed grids. Also, although most fields in ESMF have a natural physical grid associated with them, there are some cases where a physical grid is not relevant. For example a hypothetical assimilation system optimization scheme could be

Figure 9: Sketches representing various key horizontal, physical grids that ESMF will need to support. The set includes grids that are represented in functional form (for example Spherical Harmonics based grids), grids that are regular but that are typically represented as numerical values (for example the four curvilinear orthogonal grids in the right panel) and grids for which no underlying functional form exists. These latter grids are illustrated in the two lower panels in the left column. One panel (middle left) depicts a land surface grid, its grid layout typically reflects catchment basin topography, the other panel (lower left) illustrates a independent set of grid locations that is associated with a set of floats drifting in the ocean. The family of grids illustrated encompasses many other common grids that are not shown, for example cartesian and cylindrical grids.

adapted into an ESMF component. Many optimization schemes work in a linear-algebra space that is far removed from the underlying physical space, so fields that are operated on by this component might not have a realizable physical grid.

2.3 Examples

Gridding is a fundamental aspect of models supported by ESMF. Figure 9 shows a number of example horizontal grids that must be supported by ESMF. Figure ?? illustrates a subset of the vertical gridding schemes that will be supported by ESMF. For components to interoperate they will need to be able to exchange detailed information representing specific configurations of the grid types shown in the figure.

For example, under ESMF an atmospheric component employing a spectral algorithm with spherical harmonic basis functions could be coupled together with a land surface component using a catchment area based grid and with an ocean circulation component using a rotated pole grid. As discussed in the ESMF Regrid requirements chapter ??, exchanging information between these components, under appropriate conservation side-conditions, requires detailed information about the grid discretization. For this example the information that is required includes such items as

- the areas of each of the cells of the land grid and their spatial extent,

- the truncation level of the spherical harmonic series in the atmospheric model and the functional form of used to define the spherical harmonic basis functions

- the land and sea grid cell locations and areas in the ocean model associated with both terms in the momentum equations and thermodynamic terms.

Figure 9 also shows so-called “ungridded” information. Examples of “ungridded” data in ESMF derive from ingestion of observations into components. Ingesting and comparing to observational measurements entails working with grids that are associated with point wise and track measurements, for example

- an assimilation system might need to sample flow simulated in an atmospheric component in a manner consistent with radiosonde balloon network measurements or with satellite track data

- an ocean state estimation system might need to sample and analyze fields in a manner consistent with ship and satellite track or current meter observations.

In these examples both horizontal and vertical physical grid remappings may be required, in order to represent simulated data in the observational space and vice-versa.

3 Physical grid requirements

PG1 Physical locations

A mechanism shall be provided for describing physical locations in space in 1, 2, or 3 dimensions, including both specification of points and of ranges.

Priority: 1.

Source: Required by CAM-EUL, CLM, CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: System test.

PG1.1 Horizontal locations

PG1.1.1 Horizontal coordinates

Physical domains may use Cartesian, spherical, or cylindrical coordinate systems in the horizontal directions. Units for these coordinates are meters (for Cartesian), degrees of latitude and longitude (for spherical), and meters and degrees for the radius and angle in cylindrical coordinates.

Priority: 1.

Source: Required by CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, CAM-FV, PSAS, MIT, GFDL

Status: Approved-1.

Verification: Code inspection.

Notes: This suggestion follows common practice, but is an explicit exception to the requirement that MKS units are used by ESMF codes wherever units must be assumed.

PG1.1.2 Horizontal locations may be points

Horizontal locations may be specified as a pair of real values in the order (X,Y) or (longitude, latitude).

Priority: 1.

Source: Required by POP, CICE, Regrid, NCEP-GSM, NCEP-SSI, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

PG1.1.3 Horizontal locations may be polygonal regions

Horizontal locations may be specified to be regions by providing the number of vertices and the list of the vertex points. Vertex points must be specified either clockwise or counterclockwise around the region. The vertex points may be redundant.

Priority: 1.

Source: Required by POP, CICE, Regrid, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: Fundamental to allowing conservative interpolation, and for a precise description of data locations.

PG1.1.4 Horizontal regions may have central points

Both a central point and a region may be specified in describing a horizontal location. The points may provide a convenient nominal location, even when a value actually pertains to a region.

Priority: 1.

Source: Required by POP, CICE, Regrid, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: Many models mix finite difference and finite volume concepts.

PG1.1.5 Horizontal regions may be circular

A horizontal location may be specified by adding a nominal radius of influence to the central point. This may be the radius of a Gaussian distribution of influence. The exact interpretation of this radius is the responsibility of user-provided software.

Priority: 2.

Source: Required by PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

Notes: This is necessary for describing certain observational data streams.

PG1.1.6 Paths between grid locations may be specified

A method for determining the path connecting grid locations is required. This path would be used to accurately compute intersections for Regridding, lengths of cell sides, grid cell areas and a variety of other grid metrics. A linear approximation between points is a proper assumption for cartesian grids and for computing sides of latitude/longitude or reduced grid cells. A linear approximation is also adequate in many other cases and would be a logical default choice. The most accurate solution would permit users to pass a subroutine which provides analytic or highly-accurate discrete forms of the grid Jacobian (the matrix of partial derivatives of the physical coordinates with respect to logical coordinates). An additional possibility might internally support analytic forms like great circles or higher-order approximations (eg quadratic approximation to the cell side given a midpoint in addition to the two endpoints).

Priority: 1.

Source: Regrid, required by codes that use conservative interpolation.

Status: Approved-1.

Verification: Unit test.

Notes: Necessary for allowing conservative interpolation.

PG1.2 Vertical locations

PG1.2.1 Vertical coordinates

Physical domains may use a variety of vertical coordinates, including pressure, height, density, isotherms, sigma, other terrain-following, or any other vertically monotonic quantity. In addition, a user-interpretable vertical proxy (such as a satellite measurement channel) may be used. Units of this coordinate must be self-consistent. (See the CF convention for a full discussion of options for vertical coordinates at <http://www.cgd.ucar.edu/cms/eaton/netcdf/CF-20010629.htm>)

Priority: 1.

Source: Required by CAM-EUL, POP, NCEP-GSM, NCEP-SSI, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Code inspection.

PG1.2.2 Vertical locations may be points

Priority: 1.

Source: Required by CAM-EUL, POP, NCEP-SSI, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

PG1.2.3 Vertical locations may be regions

Vertical locations may be specified by providing the values of the top and bottom bounding points. Such regions have the same extent regardless of the order in which the bounding points are specified.

Priority: 1.

Source: Required by POP, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG1.2.4 Vertical regions have central points

Both a central point and a region may be specified in describing a vertical location. The points may provide a convenient nominal location, even when a value actually pertains to a region.

Priority: 1.

Source: Required by POP, NCEP-GSM, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: Many models mix finite difference and finite volume concepts.

PG1.2.5 Vertical locations may have radii of influence

A vertical location may be specified by adding a nominal radius of influence to the central point. This may be the radius of a Gaussian distribution of influence. The exact interpretation of this radius is the responsibility of user-provided software.

Priority: 3.

Source:

Status: Proposed.

Verification: Unit test.

Notes: This appears superfluous. It is more than enough to specify a range in 1D - Arlindo.

PG1.2.6 Vertical locations may include lopped cells

Vertical locations may include a region whose bounds vary between the horizontal corners of a region.

Priority: 2.

Source: Required by MIT, NSIPP, POP, GFDL/MOM.

Status: Approved-1.

Verification: Unit test.

PG2 Location streams

Streams of locations are used to describe the physical (and potentially temporal) locations associated with streams of data. Streams of locations differ from physical grids (see below) in that there are no concept of neighboring values, topology, covering a space, or of locations being exclusive.

Priority: 1.
Source: Required by NCEP-SSI, PSAS, MIT, GFDL.
Status: Approved-2.
Verification: System test.

PG2.1 Location streams may be created

Priority: 1.
Source: NCEP-SSI, PSAS, MIT, GFDL.
Status: Approved-2.
Verification: Unit test.
Notes:

PG2.2 Location streams may be destroyed

Priority: 1.
Source: NCEP-SSI, PSAS, MIT, GFDL.
Status: Approved-2.
Verification: Unit test.
Notes:

PG2.3 Location streams may be copied

Given an existing location stream, a new stream may be generated with a new name and possibly a different length.

Priority: 2.
Source: PSAS, MIT, GFDL.
Status: Approved-2.
Verification: Unit test.

PG2.4 Reading streams

Location streams may be read from files.

Priority: 1.
Source: Required by NCEP-SSI, MIT, GFDL.
Status: Approved-2.
Verification: Unit test.
Notes: I no longer find this necessary. It is sufficient to have an IO requirement for fields (which can be defined on location streams) - Arlindo. - ditto CNH (but it is a requirement!).

PG2.5 Writing streams

Location streams may be written to files.

Priority: 1.
Source: Required NCEP-SSI, MIT, GFDL.
Status: Approved-2.
Verification: Unit test.
Notes: See previous - Arlindo.

PG2.6 Background grid

A location stream may have associated with it an underlying physical grid, so that each location stream element may be uniquely associated with a single grid cell. (A single grid cell may contain multiple location stream elements.)

Priority: 1.

Source: Required by NCEP-SSI, PSAS, MIT, GFDL

Status: Approved-2.

Verification: Unit test.

Notes: This is necessary for such operations as halo updates on a location stream.

PG2.7 Location stream attributes

PG2.7.1 Fixed length location streams

Location streams may be of fixed length, specified at the time of generation.

Priority: 2.

Source: Required by PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG2.7.2 Extensible length location streams

Location streams may be of extensible length (e.g. a linked list), with an initial length specified at the time of generation.

Priority:

Source:

Status: Rejected (are we sure?? - CNH, RWH).

Verification: Unit test.

Notes: Are we sure we want to reject? Could they just be not as optimised. Imagine a location stream that is representing Lagrangian elements in a decomposed fluid simulation. It could be useful to be able to add particles as needed - CNH.

PG2.7.3 Global attributes: location stream name

Each location stream has a unique name by which it can be referred.

Priority:

Source:

Status: Rejected.

Verification: Unit test.

Notes: I think this should be deferred - CNH.

PG2.7.4 Location stream registry

Upon creation, the name and a pointer to each location stream shall be stored in a registry. A pointer to any location stream may be determined given its name.

Priority:

Source:

Status: Rejected.

Verification: Unit test.

Notes: I think this should be deferred - CNH.

PG2.7.5 Global attributes: number of dimensions

A location stream may be queried for the number of dimensions, which is set at the time of creation of the stream.

Priority: 2.

Source: Required by PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Code inspection.

PG2.7.6 Global attributes: dimension names

Each dimension has a name, which may be set and queried.

Priority: 2.

Source: Required by PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG2.7.7 Global attributes: dimension units

A location stream contains the units of each dimension, which may be set and queried.

Priority: 2.

Source: Required by PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG2.7.8 Global attributes: text or numeric attributes

A location stream may have an arbitrary number of text or numeric attributes, which may be added, set and queried. Each attribute has a text name by which it can be queried. Also, a location stream can be queried for a list of all global attribute names.

Priority: 2.

Source: Required by NCEP-SSI, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

Notes: Not quite sure what this is. Again, since we can define fields on location streams, not sure we need this here - Arlindo - ditto CNH (but it is a requirement!).

PG2.7.9 Global attributes: number of elements and number in use

The number of elements in a location stream is available. For a fixed length stream, both the total number of elements and the number of elements before the last active element location may be queried.

Priority: 2.

Source: Required by NCEP-SSI, PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG2.7.10 Global attributes: null element location value

Each location stream may indicate whether a particular location is active.

Priority:

Source:

Status: Rejected.

Verification: Code inspection.

Notes: This functionality may be obtained with user-defined element attributes, but this use will not be explicitly supported by ESMF.

PG2.7.11 Elements in stream have similar properties

All elements in a location stream will have the same numbers of dimensions, use the same physical coordinates, the same units, the same element attributes (attributes at some locations may be missing).

Priority: 2.

Source: MIT.

Status: ????

Verification: Code inspection.

Notes: MAJOR OBJECTION. Where did this come from? I'd like to be able to represent the whole observation vector for a given synoptic time on a single location stream. While data have horizontal coordinates in (lat,lon), vertical coordinates may vary widely (winds at 10m above sfc, temperature at 500 hPa, to name a few). So, each element would contain the (lat,lon,lev,levunits); it is conceivable that at some point one would need (lat,lon,lev,xunits,yunits,zunits) - Arlindo. NO STATUS ASSIGNED WE NEED TO REVISIT THIS - CNH/RWH.

PG2.7.12 Elements include values of locations

Methods shall be provided to set and query each element's location.

Priority: 1

Source: Required by PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG2.7.13 Elements may be copied

Methods shall be provided to copy all information from one element to another. When elements are copied from one location stream to another, all corresponding properties and attributes will be copied, while missing attributes will be set to missing or a user-provided default.

Priority: 1

Source: Required by PSAS, GFDL, MIT.

Status: Approved-2.

Verification: Unit test.

PG2.7.14 Elements may have attributes

Text or data attributes may be attached to each element. These attributes may be null for any particular element. Methods shall be provided to set and query each element's attribute. Element attributes may use standard names to promote interoperability.

Priority: 3

Source:

Status: Proposed.

Verification: Unit test.

Notes: I don't see the point of this as one can define fields on location streams - Arlindo.

PG2.7.15 Location streams may contain null (discarded) elements

Some elements within a location stream may be set to be invalid. This may be a way to specify the elements that are irrelevant for a particular subdomain.

Priority:

Source:

Status: Rejected.

Verification: Unit test.

Notes:

PG2.7.16 Location streams may be queried for valid elements

Location streams may be queried to obtain an ordered list of the indices of (or pointers to) all valid elements.

Priority:

Source:

Status: Rejected.

Verification: Unit test.

PG2.8 Location stream methods requiring registries of dependent data

If all of the data streams that use a particular location stream are known, additional methods for manipulating location streams and associated data streams are possible.

Priority:

Source:

Status: Rejected.

Verification: Unit test.

Notes: I don't know what this means - CNH.

PG2.8.1 Registry of data streams

Each location stream includes a registry of all the data streams that rely upon a location stream. This is necessary for location streams and data streams to be manipulated in compatible ways.

Priority:

Source:

Status: Rejected.

Verification: Unit test.

Notes: See previous req (what is a "data stream") - CNH.

PG2.8.2 Extensible location streams may be extended

Priority:

Source:

Status: Rejected.

Verification: Unit test.

Notes: Who rejected extensible location streams? Who proposed them? - CMD

PG2.8.3 Extensible location streams may be shortened

Priority:

Source:

Status: Rejected.

Verification: Unit test.

PG2.8.4 Extensible length location streams may be converted to fixed length

Priority:

Source:

Status: Rejected.

Verification: Unit test.

PG2.8.5 Fixed length location streams may be converted to extensible length

Priority:

Source:

Status: Rejected.

Verification: Unit test.

PG2.8.6 Fixed length streams may have null elements moved to end

Priority:

Source:

Status: Rejected.

Verification: Unit test.

PG3 Physical grids

A physical grid identifies a set of locations in physical space. Local physical grids provide the locations of each of the cells/points associated with the range of indices in a distributed grid. A local physical grid is associated with a single distributed grid. It may have undistributed dimensions that are not present in the underlying distributed grid. Multiple local physical grids may be derived from the same global, undistributed physical grid.

Physical grids may be purely horizontal, purely vertical, or 3-dimensional. Structured grids assume that adjacent locations in index space share boundaries in a predictable way. Unstructured grids also have concepts of neighboring cells, but the relative indices of neighbors are unpredictable.

Priority: 1.

Source: Required by CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: System test.

PG3.1 Reading grids

Given a distributed grid, a local physical grid can be read from a standard file containing a global physical grid. If no distributed grid is provided, the global physical grid will be read in.

Priority: 1.

Source: Required by CAM-EUL, CLM, CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.2 Writing grids

Physical grids can be output to standard files.

Priority: 1.

Source: Required by CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes:

PG3.3 Local physical grids may be internally generated

For an arbitrary number of points in the global domain of the associated distributed grid, it may be possible to specify an algorithm for internally determining the local physical grid.

Priority: 1.

Source: Required by POP, CICE, NCEP-GSM, NCEP-SSI, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.4 Null physical grid creation

It shall be possible to create any data objects associated with a physical grid without providing the data that a physical grid will contain.

Priority: 3

Source: Required by CCSM-CPL, MIT.

Status: Proposed.

Verification: Unit test.

PG3.5 Physical grid query.

Methods shall be provided to query a physical grid for all information in contains.

Priority: 1

Source: Required by CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.6 Cell specification

Physical grids shall specify both the locations of cell vertices, and the locations of cell centers.

Priority: 1.

Source: Required by POP, CICE, Regrid, NCEP-GSM, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: Many models mix finite difference and finite volume concepts.

PG3.7 Refinement

A physical grid may be interpolated to generate a physical grid with an equivalent span at finer or coarser resolution. Methods should be provided to accomplish such interpolation via a simple interface that uses the Regrid facility.

Priority: 1.

Source: Required by MIT(handled by regrid), GFDL(handled by regrid)

Status: Approved-1.

Verification: Unit test.

Notes: Necessary for runtime configurable resolution. Also, note that there may be a Catch-22 here, as Regrid would naturally provide the facility for Regridding, but Regrid will typically require the target physical grid for creating the Regridding. This requirement is also explicitly addressed within the Regrid requirement document.

PG3.8 Regeneration

A new local physical grid may be generated for a given distributed grid from another physical grid. The span of the source physical grid may be the same as or a superset of the span of the target.

Priority: 1.

Source: MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: Necessary for support of transposes, or of moving nests.

PG3.9 Distributed grid reference

A local physical grid may be queried for the distributed grid upon which it is based.

Priority: 2.

Source: MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG3.10 Horizontal coordinate independent of vertical

Horizontal physical grid locations can be assumed independent of the vertical coordinate. The horizontal metrics, however, may be function of the vertical coordinate, as in thick-shell spherical coordinates.

Priority: 1.

Source: Any Objections?

Status: Approved-1.

Verification: Code inspection.

Notes: If this assumption can be made, it greatly simplifies implementation. No widely used counterexamples are known.

PG3.11 Vertical coordinate potentially dependent on horizontal.

Vertical physical grid locations may be functions of the horizontal coordinates, or may be independent of them.

Priority: 1.

Source: GFDL-MOM4 (required), NSIPP, POP, CAM-FV, PSAS, MIT, GFDL

Status: Approved-1.

Verification: Code inspection.

Notes: This is necessary to support, for example, partial cells in Z-coordinate ocean models.

PG3.12 Dimension extension

A new physical grid may be generated by adding a dimension to an existing physical grid. The span of the source physical grid may be the same as or a superset of the span of the target. For a local physical grid, the new dimension will be independent of the underlying distributed grid, and both local physical grids share the same distributed grid. The new dimension may be in any order with respect to existing dimensions.

Priority: 2

Source: CCSM-CPL, GFDL, MIT

Status: Approved-2

Verification: Unit test

Notes: Valuable for separating generation of vertical and horizontal coordinates.

PG3.13 Dimension reduction

A new physical grid may be generated by removing a dimension from an existing physical grid. For a local physical grid, if the dimension that is removed is one that is present in the original underlying distributed grid, an appropriately reduced distributed grid must also be provided. Otherwise the new local physical grid is based on the same distributed grid as the original physical grid.

Priority: 2

Source: CCSM-CPL, GFDL, MIT

Status: Approved-2

Verification: Unit test

Notes:

PG3.14 Arbitrary dimensional physical grids

Physical grids may have an arbitrary number of dimensions.

Priority:

Source: ?

Status: Rejected.

Verification: Unit test.

Notes: If supported, this facility would dramatically complicate implementation, without adding much functionality.

PG3.15 1- 2- or 3- dimensional local physical grids

Local physical grids may have up to 3 dimensions, but must have at least as many dimensions as the underlying distributed grid.

Priority: 1

Source: Required by CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, CAM-FV, PSAS, MIT, WRF, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.16 Index order

A physical grid may use any index order (XYZ, XZY, etc.). Methods shall be provided to specify the order upon creation and to query the order of a physical grid. It shall also be possible (if not efficient) to extract physical grid information in any index order.

Priority: 1

Source: Please list required orders in Fortran notation. CCSM-CPL(XY), CAM-EUL (XYZ, XZY, ZXY), POP(XYZ), CICE(XY), NCEP(XYZ,XZY,YXZ,ZXY,ZYX), WRF(XYZ,XZY,YXZ,ZXY,ZYX), MIT(XYZ, XZY, ZXY, YZX), PSAS (XYZ,XZY), GFDL (XYZ, ZXY, XZY, YZX)

Status: Approved-1.

Verification: Unit test.

Notes: Necessary for support of transposes.

PG3.17 Dimension reordering

A new local physical grid may be generated with reordered dimensions from another local physical grid. If the new dimension order is inconsistent with the original distributed grid, a new consistent local physical grid must also be provided. To be consistent, all dimensions present in a distributed grid must have the same relative order in the local physical grid. (i.e. if the distributed grid uses XY, local physical grids using XYZ, ZXY, or XZY are all consistent, while one using ZYX is not.)

Priority: 2

Source: GFDL, MIT

Status: Approved-2.

Verification: Unit test.

Notes: Necessary for support of transposes.

PG3.18 Location index determination

A method shall be provided to return the cell index of a location. An option shall be provided to either create an exception for any location outside of the valid range of the coordinate system, or to produce a gracefully treatable return value if the location is (1) outside of the range of the local physical grid, or (2) outside of the range of the global physical grid. The index locations should be floating point numbers to facilitate interpolation.

Priority: 1

Source: Required by CCSM-CPL, POP, Regrid, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.19 Index location determination

A method shall be provided to return the physical locations from a physical grid of floating point index coordinates. Any index in the global physical grid may be used, although there may be performance differences between points that are on and off of the local physical grid.

Priority: 1

Source: Regrid (maybe - depending on implementation).

Status: Approved-1.

Verification: Unit test.

PG3.20 Horizontal physical grids

PG3.20.1 Physical grids map projections

Physical grids may be generated from a number of standard map projections, including traditional and Mercator grids on a sphere, rotated latitude-longitude, tripolar, and Gaussian cylindrical grids. Additional requested grids include cubed-sphere, polar stereographic, and Lambert conformal projections.

Priority: 1.

Source: Required by POP, CICE, NCEP-GSM, NCEP-SSI, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: Perhaps some of these should be read in from a file, rather than internally generated - CNH? The underlying (background?) grid associated with a location stream may be in one of these map projections - Arlindo.

PG3.20.2 Unstretched cartesian internal generation

A simple interface shall be provided to internally generate a uniform Cartesian coordinate physical grid, given the lengths of the edges of a square domain.

Priority: 1.

Source: MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.20.3 Latitude-longitude internal generation

A simple interface shall be provided to internally generate a uniform (constant grid-spacing in degrees) latitude-longitude physical grid, given the extent of the domain in latitude and longitude.

Priority: 1.

Source: CCSM-CPL, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.20.4 Stand-alone global physical grid generation examples

Stand-alone software examples shall be provided to demonstrate the generation of a global physical grid file on a stretched latitude-longitude grid, a rotated latitude-longitude grid and a tripolar grid.

Priority: 2.

Source: GFDL, NCAR, DAO, MIT.

Status: Approved-2

Verification: Unit test.

Notes: These are intended both for real use, and for use as patterns in the creation of physical grid files for more complicated grids. The above list may be altered, extended or reduced following discussions.

PG3.20.5 Supported topologies

Supported horizontal grid topologies will include logically rectangular grids that are reentrant (periodic) in 0, 1, or 2 directions, northern and southern tripolar (Murray 1996), sphere, icosahedral, and unstructured grids. Unstructured arrays of logically rectangular grids [for cubed-sphere (Rancic et al. 1996), reduced grids, and arbitrary nesting] will also be supported.

Priority: 1.

Source: Required by POP, CICE, NCEP-GSM, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

Notes: Topologies are intrinsic to both distributed grids and physical grids. Since the topology information is so widely used in distributed grids, and since distributed grids are used to initiate local physical grids, it is perhaps reasonable to make topology a property of a distributed grid, which is then inherited and checked by a local physical grid.

PG3.20.6 Local physical grid topology consistency checking

A mechanism shall be provided to verify that the locations of the points in a local physical grid are consistent with the topology of the underlying distributed grid. An exception shall be generated in case of inconsistency.

Priority: 1.

Source: Required by CCSM-CPL, POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.20.7 Areas tile sphere

It may be specified that grid areas should be calculated using algorithms that guarantee that the grid exactly (algorithmically to within 1 part in 10^{12}) tiles the sphere (or a portion of it).

Priority: 2.

Source: GFDL (required), Regrid, CAM-FV, PSAS, MIT.

Status: Approved-2.

Verification: Unit test.

Notes: Needed to permit exact conservation of fluxes between models.

PG3.20.8 Staggered grids

Staggered grids will be supported as a single physical grid.

Priority: 1.

Source: Required by POP, CICE, CAM-FV, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Code inspection.

Notes: Standard requirement of a staggered grid.

PG3.20.9 Available subgrids

For locally quadrilateral horizontal grids, information shall be available for each of the 4 related subgrids. That is if a t-cell is centered at a tracer point, cells centered on the east face, north face, and northeast corner of the t-cell will also be provided in the case of a NorthEast underlying distributed grid.

Priority: 2.

Source: GFDL, MIT (required), POP, CICE.

Status: Approved-1.

Verification: Code inspection.

Notes: Standard requirement of a staggered grid.

PG3.20.10 Extensible grid point representations

It is not anticipated that all possible grids will be included in ESMF. It must, therefore, be relatively straightforward to add new grids to the framework and to share those grid "extensions" amongst the framework community. For example it should be possible to add an icosahedral grid.

Priority: 2

Source: Required by POP(future icosahedral), CICE(future), PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Code inspection.

Notes: This is basic to the extensibility of ESMF.

PG3.21 Horizontal functional representations

A spectral horizontal description may be used. More generally, the horizontal structure of information may be given by specifying functional decompositions.

Priority: 2
Source: MIT, GFDL.
Status: Approved-2.
Verification: Unit test.

PG3.21.1 Horizontal Fourier grids

Cartesian Fourier grids will be supported. Associated with this grid are the wavenumbers (in units of m^{-1}) of each of the elements on the grid.

Priority: 3
Source: GFDL (desired)
Status: Proposed.
Verification: Unit test.

PG3.21.2 Horizontal spherical harmonics grids

Spherical harmonics grids will be supported. Associated with this grid are the wavenumbers (nondimensional m,n) of each of the elements on the grid. At a minimum, rhomboidal and triangular truncations will be supported.

Priority: 1.
Source: Required by CCSM-CPL, CAM-EUL, NCEP-GSM, NCEP-SSI, GFDL.
Status: Approved-1.
Verification: Unit test.

PG3.21.3 Mixed physical and Fourier grids

Mixed physical and Fourier grids will be supported. In particular, a grid on the sphere that is latitude in one dimension and Fourier zonal wavenumber (nondimensional m) in the other dimension will be supported.

Priority: 1
Source: Required by NCEP-SSI, NCEP-GSM, GFDL.
Status: Approved-1.
Verification: Unit test.

PG3.21.4 Extensible horizontal functional representations

The physical grid design should not preclude the user from using alternative functional horizontal representations, such as spectral elements.

Priority: 3
Source:
Status: Proposed.
Verification: Code inspection.
Notes: This is basic to the extensibility of ESMF.

PG3.22 Vertical functional representations

PG3.22.1 Vertical user defined functions

The physical grid design should not preclude the user from using functional vertical representations, such as EOFs, eigenfunctions, and finite elements. The vertical coordinate of such a grid might be a wavenumber or a similar quantity. (???THIS PART MAY BE MORE APPROPRIATE FOR Regrid...) Support will provided for accepting a user supplied matrix or function that would transform the function into some vertical physical space. Regridding would then be able to perform the transform, the inverse transform, and the adjoint transform.

Priority: 1.

Source: NCEP-SSI.

Status: Approved-1.

Verification: Code inspection.

PG3.23 Area overlap checking

A method shall be provided to check that physical grid cells do not overlap.

Priority: 2.

Source: Required by GFDL, CCSM-CPL, Regrid, MIT.

Status: Approved-2.

Verification: Unit test.

Notes: Standard self-consistency test.

PG3.24 Physical grid attributes

PG3.24.1 Physical grid name

Each physical grid has a unique name by which it can be referred. If no name is specified, one will automatically be generated.

Priority: 2

Source: Required by CCSM-CPL, MIT.

Status: Approved-2.

Verification: Unit test.

PG3.24.2 Number of dimensions

A physical grid may be queried for the number of dimensions, which is set at the time of its creation. Corresponding local and global physical grids have the same number of dimensions.

Priority: 1.

Source: Required by CCSM-CPL, Regrid, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.24.3 Dimension names

Each dimension has a name, which may be set and queried. If no name is specified for a dimension, a name will be automatically generated.

Priority: 1.

Source: CCSM-CPL, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.24.4 Dimension lengths

A physical grid may be queried for the local or global lengths of each of its dimensions.

Priority: 1.

Source: Required by CCSM-CPL, Regrid, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.24.5 Dimension attributes and units

A physical grid contains the units of each dimension, which may be set and queried. Dimensions may also have additional named attributes.

Priority: 1.

Source: Required by CCSM-CPL, Regrid, PSAS, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG3.24.6 Global attributes

A physical grid may have an arbitrary number of text or numeric attributes, which may be added, set and queried. Each attribute has a text name by which it can be queried. Also, a physical grid can be queried for a list of all global attribute names.

Priority: 2.

Source: Required by CCSM-CPL, PSAS, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG4 Grid metrics

Grid metrics are all of the lengths (or partial derivatives of distances with index number) and related quantities required to do a variety of calculations. All metrics are a function of the grid and must be static with time. Metric-like fields that vary with time (thicknesses in isopycnal/isentropic coordinates or node locations in fully Lagrangian codes) are not handled by the physical grid.

Priority: 1

Source: Required by POP, CICE, MIT, GFDL.

Status: Approved-1

Verification: System test.

PG4.1 Calculation of metrics

All metrics may be calculated from grid locations.

Priority: 1.

Source: Required by MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG4.2 Reading metrics

All metrics may be read from a standard grid file.

Priority: 1.

Source: Required by POP, CICE, MIT, GFDL.

Status: Approved-1.

Verification: Unit test.

PG4.3 MKS metric units

Metrics have units of m or m^2 , or other appropriate MKS units.

Priority: 1.

Source: Standard MKS requirement?

Status: Approved-1.

Verification: Code inspection.

Notes: Does MKS allow things like Pascals? or is that SI - CNH. Some metrics may not have units i.e. scale factors with latitude etc... - CNH.

PG4.4 Available metrics

All metrics are optional. In a particular instance of a physical grid it shall be possible to specify which metric terms are available. Typically, available metric information includes an extensive list of grid lengths, cell areas, and the angle between logical and physical north. Some functionality (e.g. certain Regridding) may be limited if certain common metric terms are omitted.

Priority: 1.

Source: POP, CICE, MIT, GFDL.

Status: Approved-1.

Verification: Code inspection.

Notes: All models require some subset of this information.

PG4.5 On-demand metrics

In cases where one metric can be generated internally either from grid information or from other metrics or from another physical grid, a method may be provided to create that metric field only once it is clear that it will be needed.

Priority: 3

Source: POP(desired), MIT, GFDL.

Status: Approved-2.

Verification: Code inspection.

Notes: With extensive metric information, this may be necessary to save space.

PG4.6 Query by name

It shall be possible to query for a reference to a metric field by name.

Priority: 1

Source: MIT, GFDL

Status: Approved-1.

Verification: Unit test.

PG4.7 Standard metric naming convention

A standard metric naming convention will be specified or established to facilitate the widespread use of metric information. Individual applications need not follow this convention, but may not achieve full functionality without it.

Priority: 2
Source: GFDL, MIT
Status: Approved-2.
Verification: Code inspection.

PG4.8 Dimensionality of metrics

In cases where metric terms are independent of one or more dimensions, they may be stored in arrays that omit those dimensions.

Priority: 2
Source: GFDL/HIM, POP, CICE, MIT.
Status: Approved-2.
Verification: Unit test.
Notes: This may be necessary for adequate cache/register performance. This may need to be done at compile time?

PG4.9 Available structured horizontal quadrilateral grid metrics

Available metric information may include an extensive list of grid lengths, cell areas, and the angle between logical and physical north.

Priority: 1.
Source: GFDL, POP, CICE, MIT.
Status: Approved-1.
Verification: Code inspection.
Notes: All using quadrilateral horizontal models require some subset of this information.

PG4.9.1 Cell areas

Cell areas may be available for each of the 4 related subgrids.

Priority: 1.
Source: GFDL (required), POP(some), CICE(some), MIT.
Status: Approved-1.
Verification: Code inspection.

PG4.9.2 Half-edge lengths

Each of the 8 half-edge lengths may be available for each of the 4 related subgrids. Since neighboring cells share edges, it is desirable (although it violates the proposed CF convention) to include only 4 fields.

Priority: 2.
Source: GFDL/MOM (required), MIT.
Status: Approved-2.
Verification: Code inspection.

PG4.9.3 Center-to-edge distances

Each of the 4 center to edge distances may be available for each of the 4 related subgrids.

Priority: 2.

Source: GFDL/MOM (required), MIT.

Status: Approved-2.

Verification: Code inspection.

PG4.9.4 Full-edge lengths

Each of the 4 edge lengths may be available for each of the 4 related subgrids. Since neighboring cells share edges, it is desirable (although it violates the proposed CF convention) to include only 2 fields.

Priority: 2

Source: GFDL/HIM (required), POP, CICE, MIT

Status: Approved-2

Verification: Code inspection.

PG4.9.5 Edge-to-edge distances

Both of the cell edge to edge distances may be available for each of the 4 related subgrids.

Priority: 2.

Source: GFDL/HIM (required), MIT.

Status: Approved-2.

Verification: Code inspection.

PG4.9.6 Center-to-corner distances

Each of the 4 center to corner distances may be available for each of the 4 related subgrids.

Priority: 3.

Source: MIT.

Status: Proposed.

Verification: Code inspection.

Notes: This is used in some E-grid implementations.

PG4.9.7 Cell orientation

The angle at the cell center between logical and physical north may be available for each of the 4 related subgrids.

Priority: 2.

Source: GFDL (required), POP, CICE, Regrid, MIT.

Status: Approved-2.

Verification: Code inspection.

Notes: This is required for almost any non-latitude-longitude grid.

PG4.10 Available unstructured horizontal grid metrics

Available metric information may include cell areas, edge lengths, and distances between adjacent cell centers.

Priority: 2.

Source: Land Model?

Status: Approved-2.

Verification: Code inspection.

Notes: All using unstructured horizontal grids require some subset of this information.

PG4.11 Vertical metrics

Available metric information may include spacing between cell centers and faces, in units consistent with the vertical coordinate.

Priority: 1.

Source: Required by all.

Status: Approved-2.

Verification: Code inspection.

Notes: All models require some subset of this information.

PG4.12 Cell volumes

Available metric information may include 3-D cell volumes (or masses). This is not intended for use with models for which this quantity varies with time.

Priority: 1.

Source: GFDL/MOM, MIT (required), POP

Status: Approved-2.

Verification: Code inspection.

PG4.13 Methods for calculating metrics

Metrics may be calculated by either standard or user-provided algorithms. The following subrequirements provide a partial list of such algorithms, which may be augmented later.

Priority: 2.

Source: Required by POP, CICE, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

Notes: This is basic to the extensibility of ESMF.

PG4.13.1 Jacobian metric calculation

Metrics may be calculated based on user-provided grid Jacobians [the matrix of partial derivatives of the physical coordinates with respect to logical coordinates (i.e. index space)], either in discrete form or as a series of function pointers.

Priority: 3

Source: Regrid (desired).

Status: Proposed.

Verification: Unit test.

PG4.13.2 Spline metric calculation

Metrics may be calculated by discrete estimates of the grid Jacobians based upon the discrete grid locations.

Priority: 2
Source: Regrid.
Status: Approved-2.
Verification: Unit test.

PG4.13.3 Distance-based metric calculation

Metrics may be calculated from distances (Great Circle on a sphere) between the point locations of a physical grid.

Priority: 3.
Source:
Status: Proposed.
Verification: Unit test.
Notes: This is basic to the extensibility of ESMF.

PG4.14 Additional metrics

It shall be possible for a user to specify additional metric terms to be associated with a physical grid.

Priority: 2.
Source: CCSM-CPL, MIT, GFDL.
Status: Approved-2.
Verification: Unit test.
Notes: This is basic to the extensibility of ESMF.

PG5 Grid masks

Grid masks are logical arrays on a grid that indicates whether the various points on the grid are a part of a physically similar subdomain. For example, masks are used to indicate which points are a part of the ocean and which are land. Masks are also important for nested applications.

Priority: 1.
Source: Required by POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF, GFDL.
Status: Approved-2.
Verification: System test.

PG5.1 Arbitrary number of masks

A physical grid may have an arbitrary number of masks associated with it.

Priority: 2.
Source: Required by POP, MIT, GFDL.
Status: Approved-2.
Verification: Code inspection.

PG5.2 Mask names

Each of the masks associated with a physical grid is associated with a unique name. A method shall be specified to return a pointer to a mask given its name.

Priority: 2.
Source: Required by MIT, GFDL.
Status: Approved-2.
Verification: Unit test.

PG5.3 Category masks

Masks may have an arbitrary number of categories. (e.g. 1 for points in the Atlantic, 2 for the Pacific, 3 for the Mediterranean, etc.)

Priority: 2.

Source: POP, CICE, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG5.4 Multiplicative masks

Masks may consist only of the values 0 or 1, for multiplicative masking.

Priority: 2.

Source: Required by POP, MIT, GFDL.

Status: Approved-2.

Verification: Unit test.

PG5.5 Mask complement

A method shall be provided to generate the complement of a mask.

Priority: 3.

Source: MIT, GFDL.

Status: Proposed.

Verification: Unit test.

Part VI

Infrastructure Fields and Grids: Distributed Grids

1 Authors, target codes and review team

Authors: V. Balaji, Will Sawyer, John Michalakes

Review Date: 7 May, 2002

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CAM-EUL, CLM	Bettge
CAM-FV, PSAS	da Silva, Sawyer
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell

Other Reviewers: DeLuca, Neckels, Larson, Jacob

2 Distributed grid background

3 Background

Scalable implementations of finite-difference codes are generally based on decomposing the model domain into sub-domains that are distributed among processors. These domains may then be obliged to share data at their boundaries if data dependencies are merely local, or may need to acquire information from the global domain if there are extended data dependencies, as in the spectral transform, or in elliptic solvers. The *distributed grid* is a category within ESMF used for expressing, and performing operations that involve, dependencies among data distributed across processors.

3.1 Scope

The discrete representation of data fields within a model component begins with the definition of *physical grid* associated with the data. The data fields thus get defined as arrays, which are then distributed among processors. The indices associated with array locations in each dimension thus define a global *index space*. The distributed grid encompasses all the ESMF infrastructure operations associated with the index-space representation of data. Operations requiring knowledge of actual physical locations and distances between locations belong to the physical grid, and are found in the documents associated with it.

3.2 Location

The gridded component, physical grid and distributed grid elements form a closely-linked conceptual chain. The distributed grid also provides the interface to the machine layer where the scheduling, communication and memory management primitives reside. Applications should rarely need to reach beyond the distributed grid layer for direct invocation of the communication primitives.

3.3 Summary

A gridded component is associated with one or more global physical grids. Distributing a global physical grid across PEs generates a physical grid element, which is associated with a single distributed grid element, distributed across some or all of the PEs associated with the component. The distributed grid has operations to define domain decompositions on the pelist, and given user-specified data dependencies, it can define topologies, i.e. connectivities on the pelist for scheduling communication. It contains all the operations for sharing data according to those dependencies. This includes familiar operations like the halo update and data transpose. It can perform global reductions (sum, max, min, maxloc, minloc) on distributed data. The sum has a bitwise exact option. It can create a copy of the global data on one or more PEs, and scatter a global array across a PELIST.

The distributed grid will perform sign flips, vector component interchanges, and redundancy checks as needed on certain types of grids (e.g tripolar grid and cubed-sphere).

Operations within the distributed grid do not include those which are potentially dependent on grid metrics. For instance, certain kinds of averaging operations are extremely simple on regular grids. These are metric-dependent on irregularly spaced grids, and are hence considered to belong to the physical grid.

4 Distributed grid requirements

This part covers all the requirements for defining, querying and relating grid distributions or domain decompositions. The PElist and the global domain are assumed retrievable (from **Control** and **GriddedComponent** respectively). Here we set down requirements that apply to all grids. The requirements specific to certain grid types follow in Part VII.

DG1 Grid definition

DG1.1 Generation of a layout

Given a global domain and a PElist, there will be methods to derive an appropriate layout of PEs in multiple dimensions.

Priority:

Source: Required by MIT.

Status: Proposed.

Verification: Unit test.

DG1.1.1 Subdivide a layout

Subdivide a layout. All distributed grid operations shall work on such a sub-layout.

Priority:

Source: Required by NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

DG1.2 User-specified layout

It will be possible to override the internally generated layout above by direct user specification.

Priority: 3.

Source: Required by MIT.

Status: Proposed.

Verification: Unit test.

DG1.3 1D decomposition

ESMF permits 1D domain decomposition, i.e. the distribution of a (possibly multidimensional) array over a 1D layout.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

Notes: probably can be treated as a special case of 2D with no loss of generality or performance.

DG1.3.1 Index order

1D domain decomposition may apply to any of the three spatial dimensions, on data arrays specified in any index order (XYZ, XZY, etc).

Priority:

Source: Get detailed list of sources here for different index orders: full generality is potentially a lot of work. CAM-FV: (XY, XYZ, XZY). Consistently decomposed blockwise in Y.

CCSM-CPL: (XY)

NCEP-GSM: spectral (YXZ) decomposed cyclically on X; grid (XZY) decomposed cyclically on Y

NCEP-SSI: spectral (background error) (T,YX,ZT) decomposed on X; spectral (transform) (T,YX,ZT) decomposed on ZT; grid (transform) (T,Y,X,ZT) decomposed on ZT; ungridded obs (T,YX,ZT) decomposed variable blocked on YX

MIT: (XYZ) blocked along X or Y.

WRF: (XY, YX, XYZ, YZX, XZY, ZXY)

Status: Proposed.

Verification: Unit test.

Notes: In CAM-FV dynamics, 4D tracer areas are XYZT or possibly XZTY, where T is the tracer dimension. In the physics there are cases of T in the second dimension. The CAM chunking mechanism brings up several additional indexing orders (needs to be documented by other teams)

In the NCEP-SSI, tracers (T) are split into 2 different dimensions.

DG1.4 2D decomposition

ESMF permits 2D domain decomposition, i.e. the distribution of a (2- or more dimensional) array over a 2D layout.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

Notes: Should NOT be treated as a special case of 3D! CAM-FV has been extended for 2D decomposition, but it is not used by default. A region concept is required e.g. decomp. could be blocked in X and Y and regions. Regions are always separated tiles.

DG1.4.1 Index order

2D domain decomposition may apply to any two of the three spatial dimensions, on data arrays specified in any index order (XYZ, XZY, etc).

Priority: 1.

Source: Get detailed list of sources here for different index orders: full generality is potentially a lot of work.

CAM-FV (XYZ, XZY, XYZT, perhaps XZTY, others for chunked physics); The decomposition is blockwise in X-Y for physics and blockwise in Y-Z for dynamics.

CCSM-CPL: (XY)

POP (XY), CICE (XY)

NCEP-GSM: spectral (semi-implicit) (ZYX) decomposed blocked on Y and X; spectral (transform) (YXZ) decomposed cyclically on X and blocked on Z; grid (transform) (XYZ) decomposed cyclically on Y and blocked on Z; grid (physics) (ZXY) decomposed cyclically on X and Y.

NCEP-SSI: grid (T,Y,X,ZT) decomposed variable blocked on Y and X

MIT: XYZ blocked in X, Y or XY, variable sizes and holes

WRF: (XY, YX, XYZ, YZX, XZY, ZXY)

Status: Proposed.

Verification: Unit test.

DG1.4.2 1D distributed arrays associated with a 2D decomposition

Within a 2D decomposition, it must be possible for arrays to be distributed along a single axis of distribution.

Priority: 1.

Source: Required by CAM-FV (2D decomposition), NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, MIT.

Status: Proposed.

Verification:

Notes: i.e given a 2D decomposition in (i, j) , it must be possible to have arrays that are functions of (i, j) , (i, j, k) , etc but also (j, k) . CAM-FV: needed for 2D decomposition, which not the default.

DG1.5 3D decomposition

ESMF permits 3D domain decomposition, i.e. the distribution of a (3- or more dimensional) array over a 3D layout.

Priority:

Source: Anyone?

Status: Proposed.

Verification:

Notes: 4D not foreseen for CAM-FV at this time.

DG1.6 Generation of domain decomposition

Given a global domain and a layout, there will be methods to assign each point in the domain to one PE for computation.

Priority:

Source: Requirements for specific flags to control the decomposition must be added here. CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification:

Notes: In multi-threaded applications, the PE assignment is done only for each “MPI process”. Additional threaded PEs work on partitions within these computational domains.

The partitions do not necessarily have a PE associated with it. The partitions may define a task queue from which PEs assigned to threads pop tasks. The pool of PEs available for threading is drawn from the associated PE list, of which not all are part of the layout.

Threads can be mapped on to an addressable node, which can consist of multiple PEs. The main thing I’m trying to arrive at is that the partitioning into threads is independent of assignment to PEs. Principally, one must be able to define an arbitrary number of partitions organized into a queue, off which shared-memory execution threads can pop tasks as resources get freed. - VB

Thread partitioning (and all PEs) should be included in the layout and not a separate partitioning specification. This will make queries simpler and the programming interface across a variety of machines more consistent. - CMD.

DG1.7 User-specified domain decomposition

It will be possible to override the internally generated domain decomposition above by direct user specification of domain extents.

Priority: 1.

Source: CAM-FV, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

Notes: WS: there will there be a fairly wide spectrum of decomposition create methods, with varying degrees of user specification, e.g. less specific “distributed a 3-D XYZ array over a 2-D layout in a block-cyclic manner”), or more user-specified (“map the global domain to the following PE and local indices”).

DG1.8 Domain masks

It shall be possible to mask domains from the computation: i.e generate a domain decomposition where one or more exclusive domains are assigned to no PE.

Priority:

Source: POP, CICE , MIT.

Status: Proposed.

Verification: Unit test.

Notes: Possibly useful in an ocean model when a decomposition of the sphere yields entire exclusive domains containing only land points.

Not the same mask as in Req. DG10.2.1.

DG1.9 Generation of grid topology

Given the data dependencies of the numerics, ESMF will be capable of computing the connectivities required for data sharing and synchronization of exclusive domains across a distributed grid.

Priority: 1.

Source: CAM-FV, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test for specific cases.

Notes:

DG1.10 Validity of grid topology

Given a data dependency specification, ESMF will be capable of asserting whether or not a given distributed grid topology is conformant with that pattern.

Priority: 2.

Source: Required by MIT.

Status: Proposed.

Verification: Unit test for specific cases.

Notes: This would be very useful. Why doesn't anybody want it? - CNH

DG1.11 Periodic boundary conditions

ESMF will treat periodic boundary conditions along any spatial axis as a particular kind of topological feature. It will be possible to specify this as input information for grid topology generation.

Priority: 1.

Source: CAM-FV, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT, WRF

Status: Proposed.

Verification: Unit test, code inspection.

Notes: It is natural to treat this as a distributed grid feature, rather than having separate edge detection code in applications. There is more than one way to treat LBCs in a distributed grid: these will be spelt out anon.

DG2 Grid information retrieval

DG2.1 Exclusive domain retrieval

DG2.1.1 Domain extents

It shall be possible to retrieve the size of a exclusive domain along each axis of decomposition.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

DG2.1.2 Domain begin and end indices

It shall be possible to retrieve the beginning and ending indices of an exclusive domain along each axis of decomposition.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT, WRF

Status: Proposed.

Verification: Unit test.

DG2.1.3 Domain index list

It shall be possible to retrieve the full index list of an exclusive domain along each axis of decomposition.

Priority: 1.

Source: NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes:

DG2.1.4 Maximum domain extent

It shall be possible to retrieve the maximum size along each axis of decomposition of all the exclusive domains in the distributed grid.

Priority: 1.

Source: CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes:

DG2.1.5 Exclusive domain list

ESMF methods will be provided to retrieve the list of exclusive domains associated with each PE of a distributed grid.

Priority:

Source: CCSM-CPL, POP, MIT

Status: Proposed.

Verification:

Notes: Besides being able to retrieve one's own domain information, the full list of domains is also required for certain operations.

DG2.2 Local domain retrieval

DG2.2.1 Domain extents

It shall be possible to retrieve the size of a local domain along each axis of decomposition.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

DG2.2.2 Domain begin and end indices

It shall be possible to retrieve the beginning and ending indices of a local domain along each axis of decomposition.

Priority: 1.

Source: Required by CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT, WRF.

Status: Proposed.

Verification: Unit test.

DG2.2.3 Domain index list

It shall be possible to retrieve the full index list of a local domain along each axis of decomposition.

Priority: 1.

Source: Required by NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

DG2.2.4 Maximum domain extent

It shall be possible to retrieve the maximum size along each axis of decomposition of all the local domains in the distributed grid.

Priority: 1.

Source: Required by CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

DG2.2.5 Index translation for globally non-conformant local domains

Where the local array allocation is not globally-conformant, ESMF will have methods to translate the local indices to globally-conformant indices.

Priority:

Source: POP, CICE, MIT (desired).

Status: Proposed.

Verification:

Notes: It is possible (using array lower bound specifications) to generate a *globally conformant* array allocation: such that if Madras is point (80,13) in the global grid, it will be (80,13) on the local domain of any decomposition. Where this is not done, there must be methods to translate local indices to globally conformant indices. CAM-FV should be globally conformant. This would be a useful option, but its not required by current MIT code - CNH.

DG2.2.6 Local domain list

ESMF methods will be provided to retrieve the list of local domains associated with each PE of a distributed grid.

Priority:

Source: CCSM-CPL, POP, CICE, MIT.

Status: Proposed.

Verification:

Notes: Besides being able to retrieve one's own domain information, the full list of domains is also required for certain operations.

DG2.3 Memory domain retrieval

DG2.3.1 Domain extents

It shall be possible to retrieve the size of a memory domain along each axis of decomposition.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

DG2.3.2 Domain begin and end indices

It shall be possible to retrieve the beginning and ending indices of a memory domain along each axis of decomposition.

Priority: 1.

Source: Required by CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT, WRF.

Status: Proposed.

Verification: Unit test.

DG2.3.3 Domain index list

It shall be possible to retrieve the full index list of a memory domain along each axis of decomposition.

Priority: 1.

Source: Required by NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

DG2.3.4 Maximum domain extent

It shall be possible to retrieve the maximum size along each axis of decomposition of all the memory domains in the distributed grid.

Priority: 1.

Source: Required by CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

DG2.3.5 Memory domain list

ESMF methods will be provided to retrieve the list of memory domains associated with each PE of a distributed grid.

Priority:

Source: CCSM-CPL, POP, CICE, MIT.

Status: Proposed.

Verification:

Notes: Besides being able to retrieve one's own domain information, the full list of domains is also required for certain operations.

DG2.4 Global domain retrieval

DG2.4.1 Domain extents

It shall be possible to retrieve the size of a global domain along each axis of decomposition.

Priority:

Source: CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

Notes: CAM-FV: the size of the global domain is known a priori and does not need to be retrieved.

DG2.4.2 Domain begin and end indices

It shall be possible to retrieve the beginning and ending indices of the global domain along each axis of decomposition.

Priority:

Source: CCSM-CPL, POP, CIC, MIT, WRF.

Status: Proposed.

Verification: Unit test.

Notes: CAM-FV: the size of the global domain begin and end indices are known a priori and does not need to be retrieved.

DG2.5 Layout retrieval

ESMF methods will be provided to publish and retrieve the layout associated with a distributed grid.

Priority:

Source: CCSM-CPL, MIT.

Status: Proposed.

Verification: Unit test.

DG2.6 Grid topology retrieval

ESMF methods will be provided to retrieve the network of connectivities established by the data dependency patterns of the distributed grid.

Priority:

Source: CCSM-CPL, MIT

Status: Proposed.

Verification: Unit test.

Notes:

DG2.7 Which PE is a point on?

ESMF methods will be provided to query the PE associated with (i.e "owning" the exclusive domain containing) any point in a distributed grid.

Priority:

Source: CCSM-CPL, POP, MIT, WRF

Status: Proposed.

Verification:

Notes: WS: which PE is it on and (optionally?) what is its local index. Also: if one is given the PE and the local index, it should be possible to get the global index. Is it a requirement that these mappings be highly efficient (a challenging programming task)?

DG2.8 Cross-component queries

It shall be possible to retrieve the distributed grid associated with any component in an application in a form that permits all the query operations listed above.

Priority:

Source: CCSM-CPL, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Application here means "ESMF application"? - CNH

DG3 Grid relations

DG3.1 Equality of global domains

It shall be possible to state if two distributed grids share the same global domain.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT

Status: Proposed.

Verification: Unit test.

DG3.2 Equality of domain decomposition

It shall be possible to state if two distributed grids sharing a global domain are identically decomposed.

Priority:

Source: CCSM-CPL, MIT

Status: Proposed.

Verification: Unit test.

DG3.3 Equality of PE assignment

It shall be possible to state if two distributed grids with identical domain decompositions assign their exclusive domains to the identical sequence of PEs.

Priority:

Source: CCSM-CPL, POP, CICE, MIT.

Status: Proposed.

Verification: Unit test.

DG4 Halo update

DG4.1 Unblocked halo update

ESMF shall provide an unblocked halo update: where the underlying data transfer may not be complete when the call returns.

Priority: 1.

Source: CAM-FV, NSIPP, NCEP-SSI, MIT, WRF

Status: Proposed.

Verification:

Notes: Maybe we're pre-empting design issues, but I'd want issues like wait-for-completion dealt with within distributed grids.

WRF: This is only useful if we also have a way of computing separately the interior (those points that do not have data dependencies on the halo) and boundary (those around the edge that do) of an exclusive domain.

DG4.2 Blocked halo update

ESMF shall provide a blocked halo update: where the underlying data transfer is complete when the call returns.

Priority:

Source: CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT, WRF

Status: Proposed.

Verification: Unit test.

Notes: Doesn't necessarily need separate call, same as unblocked + wait-for-completion.

WRF: This should be default, and unblocked should be separate call.

DG4.3 Wait for completion

ESMF shall provide a method to wait for the completion of a previously issued unblocked halo update.

Priority: 1.

Source: CAM-FV, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

DG4.4 Validation and invalidation of halo points

ESMF shall provide methods to declare halo points valid or invalid (i.e requiring a halo update prior to use).

Priority:

Source: POP (desired), CICE (desired), MIT.

Status: Proposed.

Verification:

Notes: This feature can be used to detect when to perform a halo update. This may also be used to reduce the frequency of halo updates on high-latency networks, by declaring wide halos and performing redundant computations within them. See [3] for details. Partial updates may not necessarily support arbitrary lists of halo points, but the following are useful:

- "Update N and E halos only, including the NE corner."
- "Update only 1 row/column in each direction even if the halo width is 2".
- "Update only the outer row/column of a 2-width halo."

DG4.5 Arrays of derived type

All relevant operations in this Section DG4 must apply to data arrays where each array element is a derived data type.

Priority:

Source:

Status: Proposed.

Verification:

Notes: I hope this isn't required, but see Section DG20 below. Why did we reject this - seems like it should be deferred - CNH.

DG4.6 Adjoint of halo

A "transpose" form (in the mathematical sense of the word) is required for all the permutations that a halo can perform.

Priority: 1.

Source: MIT.

Status: Proposed.

Verification: Unit test.

DG5 Data transpose

DG5.1 Unblocked data transpose

ESMF shall provide an unblocked data transpose: where the underlying data transfer may not be complete when the call returns.

Priority: 1

Source: CAM-FV (2D decomposition), NSIPP, NCEP-GSM, NCEP-SSI, MIT, WRF

Status: Proposed.

Verification:

Notes: Maybe we're pre-empting design issues, but I'd want issues like wait-for-completion dealt with within distributed grids.

DG5.2 Blocked data transpose

ESMF shall provide a blocked data transpose: where the underlying data transfer is complete when the call returns.

Priority: 1.

Source: Required by CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification:

Notes: Doesn't necessarily need separate call, same as unblocked + wait-for-completion.

DG5.3 Wait for completion

ESMF shall provide a method to wait for the completion of a previously issued unblocked data transpose.

Priority: 1.

Source: CAM-FV (2D decomposition), NSIPP, NCEP-GSM, NCEP-SSI, MIT, WRF

Status: Proposed.

Verification: Unit test.

DG5.4 Arrays of derived type

All relevant operations in this Section DG5 must apply to data arrays where each array element is a derived data type.

Priority:

Source:

Status: Rejected – arrays only should suffice.

Verification: Unit test.

Notes: I hope this isn't required, but see Section DG20 below.

DG5.5 Adjoint of transpose

A "transpose" form (in the mathematical sense of the word) is required for all the permutations that a transpose (in the ESMF sense) can perform.

Priority: 1.

Source: MIT

Status: Proposed.

Verification: Unit test.

Notes:

DG6 Gather

It shall be possible to create a copy on any PE of the entire global array from a distributed array.

Priority: 1

Source: All, MIT

Status: Proposed.

Verification: Unit test.

Notes:

DG6.1 Allgather

It shall be possible to create a copy on all PEs of the associated Pelist of the entire global array from a distributed array.

Priority: 1

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, NCEP-GSM, NCEP-SSI, MIT

Status: Proposed.

Verification: Unit test.

Notes:

DG6.2 Partial gather

In a 2D decomposition, it shall be possible to perform a gather along either axis.

Priority: 1

Source: CAM-FV (2D decomposition), POP, MIT

Status: Proposed.

Verification: Unit test.

Notes:

DG6.3 Adjoint of gather

A "transpose" form (in the mathematical sense of the word) is required for the matrix operator a gather represents.

Priority: 1
Source: MIT
Status: Proposed.
Verification: Unit test.
Notes:

DG7 Scatter

It shall be possible to create a distributed array across a PE list from a copy on any PE of the entire global array.

Priority:
Source: All, MIT.
Status: Proposed.
Verification: Unit test.
Notes:

DG7.1 Partial scatter

In a 2D decomposition, it shall be possible to perform a scatter along either axis.

Priority:
Source: CAM-FV (2D decomposition), NSIPP, MIT.
Status: Proposed.
Verification: Unit test.
Notes:

DG7.2 Adjoint of scatter

A "transpose" form (in the mathematical sense of the word) is required for the matrix operator a scatter represents.

Priority:
Source: MIT.
Status: Proposed.
Verification: Unit test.
Notes:

DG8 Broadcast

It shall be possible to broadcast data to all PEs in the domain decomposition.

Priority:
Source: NCEP-SSI, MIT.
Status: Proposed.
Verification: Unit test.
Notes:

DG8.1 Adjoint of broadcast

A "transpose" form (in the mathematical sense of the word) is required for the matrix operator a broadcast represents.

Priority:

Source: MIT.

Status: Proposed.

Verification: Unit test.

Notes:

DG9 Bundling

ESMF shall provide a method to bundle multiple data arrays on the same distribution for aggregate data transfer.

Priority: 1.

Source: CAM-FV, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT, WRF

Status: Proposed.

Verification: Unit test.

Notes: Note that all arrays in a bundle must share the same distribution, that's the only case where aggregate data transfer methods may be simple and efficient.

Bundling methods with similar requirements may appear in other documents (e.g **Fields**, **Regrid**, **Physical Grids**), but the methods there will likely cascade to here, as the main utility of bundles lies in aggregate data transfer methods on distributed grids. The other documents should derive an equivalent list of methods listing the ones here as source.

WRF: Should the bundling have to be specified by the application? Allow this as a hint that app can provide ESMF.

We need to work to clarify the concept of a bundle throughout this doc - right now it's pretty confusing - CMD.

DG9.1 Initiate a bundle

It must be possible to initiate any number of instances of a bundle on a distributed grid.

Priority:

Source: CAM-FV, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT

Status: Proposed.

Verification: Unit test.

DG9.2 Add an array

It must be possible to add any array on a conformant distributed grid to a bundle.

Priority:

Source: CAM-FV, POP, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Mix-and-match data types allowed? Could be tricky... if array elements have differing byte lengths.

DG9.3 Delete an array

It must be possible to delete any array from a bundle.

Priority:

Source: CCSM-CPL, MIT.

Status: Proposed.

Verification: Unit test.

DG9.4 Merge bundles

It must be possible to merge bundles sharing a distributed grid.

Priority:

Source: CCSM-CPL, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Two bundles on different distributed grids may end up on the same one after a data transpose.

DG10 Global reduction operations

DG10.1 Integer global sum

ESMF shall provide methods to compute the global sum of a distributed integer array.

Priority: 1

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT

Status: Proposed.

Verification: Unit test.

DG10.2 FP and complex global sum

ESMF shall provide methods to compute the global sum of a distributed floating-point or complex array.

Priority: 1

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT

Status: Proposed.

Verification: Unit test.

DG10.2.1 FP and complex global sum under a mask

ESMF shall provide methods to compute the global sum of a 2D-decomposed floating-point or complex array under control of a mask. The mask omits an arbitrary list of array locations from the sum.

Priority:

Source: POP, CICE, MIT

Status: Proposed.

Verification: Unit test.

Notes: Not the same mask as in Req. DG1.8.

DG10.2.2 FP and complex global sum along one axis

ESMF shall provide methods to compute the global sum of a 2D-decomposed floating-point or complex array along either axis of decomposition.

Priority: 1.

Source: CAM-FV (2D decomposition), NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, MIT

Status: Proposed.

Verification: Unit test.

Notes: Required for computing zonal means.

DG10.2.3 FP and complex bit-reproducible global sum

ESMF shall provide methods to compute the global sum of a 2D-decomposed floating-point or complex array where the result is bitwise identical on different decompositions.

Priority: 1.

Source: Trace to GR. CAM-EUL, CLM, NCEP-GSM, NCEP-SSI, MIT

Status: Proposed.

Verification: Unit test.

Notes: Possibly a compile-time option of normal (optimized) global sum. I think it should runtime selectable - CNH.

DG10.3 FP and complex global checksum

ESMF shall provide methods to compute the global checksum of a distributed floating-point or complex array.

Priority:

Source: MIT (desired)

Status: Proposed.

Verification: Unit test.

Notes: A checksum is different from a sum in that there is no data loss when the data has a dynamic range exceeding the FP or complex precision. It is done by casting all FP numbers as integers before summing.

DG10.4 Adjoints of all sums except checksum are required

DG10.5 Global maximum of integer or FP data

ESMF shall provide methods to compute the global maximum of a distributed integer or FP array.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT

Status: Proposed.

Verification: Unit test.

Notes:

DG10.5.1 Location of global maximum

The location of this maximum on the distributed grid is also retrievable.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT

Status: Proposed.

Verification: Unit test.

Notes:

DG10.6 Global minimum of integer or FP data

ESMF shall provide methods to compute the global minimum of a distributed integer or FP array.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT

Status: Proposed.

Verification: Unit test.

Notes:

DG10.6.1 Location of global minimum

The location of this minimum on the distributed grid is also retrievable.

Priority: 1.

Source: CAM-FV, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, NCEP-GSM, NCEP-SSI, MIT

Status: Proposed.

Verification: Unit test.

DG11 Blocked and unblocked collectives

It shall be possible to perform gather, scatter, broadcast, and reduction in both blocked and unblocked forms.

Priority:

Source: NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

DG12 Grid staggering

There are many possible ways to specify staggered grids. Currently grid staggering is only required for logically rectilinear grids. We use the notation of [2] to specify staggering (e.g. A-, B-, C- and D-grids) together with a directional qualifier, e.g. NE to specify a forward index offset in X and Y.

DG12.1 AGRID

ESMF will support a AGRID stencil: where the vector component array locations u_{ij} and v_{ij} are located at the cell center (i, j) .

Priority: 1.

Source: CAM-FV, CAM-EUL, CLM, CCSM-CPL, MIT, WRF (SISL, 3DVAR)

Status: Proposed.

Verification: Unit test.

Notes:

DG12.2 BGRID

ESMF will support a BGRID stencil: where the vector component array locations u_{ij} and v_{ij} are located at the NE cell corner $(i + \frac{1}{2}, j + \frac{1}{2})$. Both NE and SW flavors are supported.

Priority: 1.

Source: Required by POP, CICE, MIT.

Status: Proposed.

Verification: Unit test.

DG12.3 CGRID

ESMF will support a CGRID stencil: where the vector component array locations u_{ij} and v_{ij} are located on the N and E faces $(i + \frac{1}{2}, j)$ and $(i, j + \frac{1}{2})$ respectively. Both NE and SW flavors are supported.

Priority: 1.

Source: Required by CAM-FV, POP, MIT, WRF.

Status: Proposed.

Verification: Unit test.

DG12.4 DGRID

ESMF will support a DGRID stencil: where the vector component array locations u_{ij} and v_{ij} are located on the E and N faces $(i, j + \frac{1}{2})$ and $(i + \frac{1}{2}, j)$ respectively. Both NE and SW flavors are supported.

Priority: 1.

Source: Required by CAM-FV, MIT.

Status: Proposed.

Verification: Unit test.

DG12.5 EGRID

ESMF will support a DGRID stencil: where the vector component array locations u_{ij} and v_{ij} are (include specification of E-grid here).

Priority:

Source: WRF (MesoNH).

Status: Proposed.

Verification: Unit test.

Part VII

Requirements for Specific Grid Types

The distributed grid module will provide support the following physical grids:

DG13 Tripolar grid

The tripolar grid [7] is used to address the pole problem in ocean models, by creating a grid with two poles in the Northern Hemisphere, both over land. This creates a grid topology with a fold. Vector components experience sign reversal on crossing a fold; also some points on the vector stencil may be redundant.

(figure)

Priority: 1.

Source: POP, CICE.

Status: Proposed.

Verification: Unit test.

Notes:

DG13.1 Vector component reversal

Priority: 1.
Source: POP, CICE.
Status: Proposed.
Verification: Unit test.
Notes:

DG13.2 Redundancy enforcement

Certain locations on a staggered grid have multiple index locations for a single physical location. While these points are redundant, the data there may diverge because of FP roundoff error. ESMF will provide methods to enforce exact redundancy.

Priority: 1.
Source: POP, CICE
Status: Proposed.
Verification: Unit test.
Notes:

DG13.3 Validity of grid

It must be possible to check if the grid and fold are conformant: for instance, a southern fold with a BGRID stencil needs an extra vector row at the southern edge.

Priority: 1.
Source: POP, CICE.
Status: Proposed.
Verification: Unit test.
Notes:

DG14 Cubed-sphere grid

The cubed-sphere grid [8] is used to address the pole problem in ocean models, by creating a grid with eight weak poles, which may also be moved over land if one chooses. This creates a grid topology where vector components may need to be interchanged $((u, v) \rightarrow (v, -u))$ on crossing a cube edge. Some points on the vector stencil may be redundant.

DG14.1 Vector component interchange

ESMF will correctly treat vector components crossing faces of a cubed sphere to perform the appropriate component interchanges.

Priority: 1.
Source: MIT
Status: Proposed.
Verification: Unit test.
Notes:

DG14.2 Redundancy enforcement

Certain locations on a staggered grid have multiple index locations for a single physical location. While these points are redundant, the data there may diverge because of FP roundoff error. ESMF will provide methods to enforce exact redundancy.

Priority: 1.
Source: MIT
Status: Proposed.
Verification: Unit test.
Notes:

DG15 Spectral grid

Spectral models generally have three associated grids, one in physical space, one in Fourier space, and one in spectral (wavenumber) space. The sequence of transformations to pass data between the grids is treated at its highest level within the Regrid semantics. Within these, there is generally a specific step to redistribute data to optimize specific operations (FFTS and LTs). This section will highlight the specific requirements of these redistributions.

DG15.1 Globalize on one axis

There will be an efficient method to create an array where all the global data is available along one of the axes of distribution.

Priority:
Source:
Status: Proposed.
Verification: Unit test.
Notes: Already covered by Req. DG6.2, but highlighted here.

DG15.2 Transpose axis of globalization

There will be an efficient method to transpose an array global along one axis, into one global along another. Note that this is a matrix transpose, as opposed to the general redistribution that we've called a data transpose in DG5.

Priority:
Source:
Status: Proposed.
Verification: Unit test.
Notes: Already covered by Req. DG6.2, but highlighted here.
(MI) The NCEP codes will be able to satisfy this need with data transposes.

DG16 Exchange grid

DG17 Icosahedral grid

See [4, 6] for extensive discussions on these grids.

Priority: 3.
Source: FV, POP, CICE
Status: Deferred

Verification: Unit test.

Notes: Envisioned for FV, but not a milestone. POP, CICE currently funded to do this under CSU SciDAC effort but ESMF involvement will occur in future.

DG18 Reduced grids

Priority:

Source: NCEP-GSM

Status: Proposed.

Verification: Unit test.

Notes:

DG19 Nested grids

Nested grids are overlapping grids with certain properties that may be possible to exploit for efficiency. Many of the requirements having to do with data exchange between nested grids will appear in the physical grid and Regrid documents. Some requirements of nested grids appear here: these concern the special case of *mesh-aligned* nested grids. Nesting is defined to be mesh-aligned when every coarse grid point in the region of coverage of a fine nest is also a point on the fine grid. The following requirements apply to mesh-aligned nested grids:

DG19.1 Discrete data shift on moving nests

A method will be provided for data shifting by integral grid intervals along axes of decomposition on a distributed moving mesh-aligned nested grid.

Priority:

Source: WRF

Status: Proposed.

Verification: Unit test.

Notes: i.e, a moving mesh-aligned nested grid may support a lateral data shift: “move all points on this distributed grid 2 points south and 1 east”.

DG20 Unstructured grids and ungridded data

Unstructured grids and ungridded location streams will come associated with an underlying background grid (see ??). Requirements in this section use the background grid to define operations on unstructured grids and ungridded location streams. For the purpose of a distributed grid, unstructured grids and ungridded location streams are aligned on a single axis, each index of which is associated with a background grid cell.

DG20.1 Grid association

It shall be possible to associate a structured grid with unstructured grids and ungridded location streams.

Priority: 1.

Source: PSAS, NSIPP, NCEP-SSI (milestone), MIT(assimilation and lagrangian diagnostics)

Status: Proposed.

Verification: Unit test.

Notes: The method to generate the background grid requires location information, and belongs to the physical grid. Needed for PSAS, but not a milestone.

DG20.2 Domain decomposition

It shall be possible to apply domain decomposition to unstructured grids and ungridded location streams, using the background grid.

Priority: 1.

Source: PSAS, NSIPP, NCEP-SSI (milestone), MIT(assimilation and lagrangian diagnostics)

Status: Proposed.

Verification: Unit test.

Notes: Needed for PSAS, but not a milestone.

DG20.3 Halos and halo updates

It shall be possible to apply halos and halo update operations as defined in DG4 to unstructured grids and ungridded location streams, using the background grid.

Priority: 1.

Source: PSAS, NSIPP, NCEP-SSI (milestone), MIT(assimilation and lagrangian diagnostics)

Status: Proposed.

Verification: Unit test.

Notes: Needed for PSAS, but not a milestone.

DG20.4 Data transpose

It shall be possible to apply data transposes as defined in DG5 to unstructured grids and ungridded location streams, using the background grid.

Priority: 1.

Source: PSAS, NCEP-SSI (milestone)

Status: Proposed.

Verification: Unit test.

Notes: Needed for PSAS, but not a milestone.

Part VIII

Infrastructure Fields and Grids: Regridding

1 Authors, target codes and review team

Authors: Phil Jones, Mark Iredell, Will Sawyer

Review Date: 30 April, 2002

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CAM-EUL, CLM, CCSM-CPL	Craig
CAM-FV, PSAS	da Silva
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell, Young

Other Reviewers: DeLuca, Neckels, Jacob, Larson

2 Regrid background

Many applications in Earth system modeling require transforming fields between different grids. Such cases occur when coupling two components that utilize different grids (eg coupling atmosphere and ocean components of a climate model) or when a single component utilizes more than one grid (eg nested grids, multi-grid or spectral transforms). Fields must then be interpolated, averaged or remapped from one grid to another. We will refer to this process as *regridding*.

Many different methods for performing regriddings are required. Regridding heat or water fluxes between models requires fluxes to be regridded in a way that conserves the total energy or water in the model. In other cases, it is more important that higher-order regridding methods be used to prevent discontinuities in the interpolated fields or their gradients.

In addition to moving fields between grids, components require moving ungridded data (eg observations) to and from component grids. Transforming fields from physical space to spectral space is also a frequent operation.

The ESMF Regridding facility (Regrid) will supply necessary functions and data structures for implementing regridding operations. Appropriate functions will be supplied for all supported ESMF grids, including grids in spherical, Cartesian and cylindrical coordinates and non-gridded lists of points in those coordinates (see physical grid requirements for complete list of supported grids). A variety of methods will be supplied, including conservative, spectral transform, linear and higher-order methods. The remainder of this document outlines specific requirements for all of the supported options.

2.1 Location

Regridding and interpolation is part of the ESMF Infrastructure. It will use information from physical grids or distributed grids to compute regridding information and use distributed grids for performing data motion related to regridding. It will (presumably) be used by couplers and fields as well as within component models when necessary.

2.2 Scope

ESMF Regrid is meant to be used for interpolating or remapping data between supported ESMF grids. Support for moving fields between points on staggered grids will be included in the physical grid.

RG1 Regrid requirements

RG2 General regridding requirements

The following are general requirements for regridding operations and are in addition to the applicable general ESMF requirements (see ESMF General Requirements document).

RG2.1 Creation

Components must be able to create a regridding and initialize various time-independent regridding quantities.

Priority: 1.

Source: All codes will require this.

Status: Proposed.

Verification: Unit test.

Notes: This function will, in many cases, be computing regridding weights and initializing various communication information for performing regridding.

RG2.2 Destruction

Components must be able to destroy regriddings to free up memory.

Priority: 1.

Source: POP, CICE, CAM desired, NSIPP, CCSM-CPL, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes:

RG2.3 Query

Components must be able to query various properties of a regridding.

Priority: 2.

Source: CCSM-CPL, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Ideally, components should not need to access internal regridding fields. But it might be useful to access some aspects for error checking, optimization of data layout or renormalization. Exact query functions will be determined after design of structure is determined.

RG2.4 Change

Components should be able to change individual properties of a regridding. Examples might include adjusting regridding weights to renormalize or to adapt to dynamic area fractions (like ice fraction).

Priority: 3.

Source: CCSM-CPL, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes: In general, this should be strongly discouraged as it may affect regridding properties like conservation. The need for such a function will be determined by applications.

RG2.5 Reading

Components may be able to read regridding information from a file.

Priority: 1.

Source: POP, CICE, CAM desired, NSIPP, CCSM-CPL, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Useful if creating regriddings is time-consuming to avoid start-up costs. Also permits off-line computation of regridding.

RG2.6 Writing

Components may be able to write regridding information to a file.

Priority: 1.

Source: POP, CICE, CAM desired, NSIPP, CCSM-CPL, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Useful if creating regriddings is time-consuming - compute once and save for later re-use. Also useful for any off-line use of same regridding info.

RG2.7 Support for ESMF grids

Regridding operations must be available for all supported ESMF grids, including ungridded data. Not all regridding operations are appropriate for all grid types; restrictions will be noted in individual requirements.

Priority: 1.

Source: All codes require this.

Status: Proposed.

Verification: System test

Notes:

RG2.8 Multiple fields

Regridding of multiple fields (bundles of fields) with a single call must be supported.

Priority: 1

Source: CCSM-CPL, NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes: This should be supplied for efficiency, but may not be required to achieve functionality.

RG2.8.1 Interface requires only data arrays

An interface requiring only data arrays shall be provided. Such an interface would not require the overhead of a full field structure and is supplied for efficiency.

Priority: 2.

Source: POP, CICE, CAM desired, CCSM-CPL, NCEP-GSM, NCEP-SSI, MIT, WRF.

Status: Proposed.

Verification: Unit test.

Notes: Field info may still be used for creation of the regrid object.

RG2.8.2 Consistency of field bundles

The regridding function must check if the input field bundle and output field bundle are consistent with each other, particularly in number of fields, name of fields and grids on which the fields are placed.

Priority: 2.

Source: POP, CICE, CAM desired, NSIPP, CCSM-CPL, NCEP-SSI, NCEP-GSM, MIT.

Status: Proposed.

Verification: Unit test.

Notes: A rudimentary error check, but would probably rely on consistent naming convention for fields?

RG2.9 Multiple methods per grid pair

It shall be possible to create more than one regridding for a given grid pair.

Priority: 1.

Source: POP, CICE, CAM required, CCSM-CPL, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Both non-conservative and conservative methods will be needed between the same two grids.

RG2.10 Consistency of coordinates

Regridding will assume source and destination grids will be in compatible coordinate systems. No knowledge of the projection used or the physics of the coordinate are required for performing a regridding. For example, if a horizontal grid is in Cartesian coordinates, the second grid must also be in Cartesian coordinates with the same origin. Similarly, if the coordinates of one grid are in spherical coordinates, the second grid must also use spherical coordinates. The restriction also applies to vertical coordinates where the regridding will not be expected to know how to transform between two different coordinate choices (eg pressure to isentropic).

Priority: 1.

Source: Required by all.

Status: Proposed.

Verification: Code inspection

Notes: Exceptions to this are permitted if the user supplies the regridding routine (see later requirement on user-supplied regridding).

RG2.10.1 Consistency of coordinates check

The regridding function must check that the grids are in fact consistent with each other.

Priority: 2.

Source: Required by all.

Status: Proposed.

Verification: Code inspection

Notes: Will require some standard nomenclature for grid attributes, particularly for vertical grids.

RG2.11 Interpolation adjoints

Adjoints shall be supplied for regridding methods when possible. This is generally possible for regriddings that are independent of the field being regridded (see following requirement) and that can be cast as a linear operator (eg matrix multiplication). Methods where adjoints are absolutely required have been so noted within their own respective descriptions.

Priority: 2.
Source: PSAS, NSIPP, NCEP-SSI, MIT (milestone).
Status: Proposed.
Verification: Unit test.
Notes: Needed by PSAS, but not milestone

RG2.12 Masked regridding

It shall be possible to restrict the regridding to parts of a grid through the use of a mask. Note that use of a mask is inappropriate for some methods (eg spectral transforms) and will not be supported for those methods.

Priority: 1.
Source: CCSM-CPL (desired)
WRF required, NSIPP, NCEP-GSM, MIT.
Status: Proposed.
Verification: Unit test.
Notes:

RG2.12.1 Mask consistency

If masks are supplied for both source and destination grids, a method for checking consistency of those masks must be supplied. Alternatively, a convention for resolving mask conflicts must be determined (eg source grid is “master”).

Priority: 1.
Source:
WRF required, NSIPP, NCEP-GSM, MIT.
Status: Proposed.
Verification: Unit test.
Notes:

RG2.13 Independence of field

Whenever possible, regridding should be formulated to be independent of the field being regridded. This requirement exists to aid the creation of an adjoint, to enable pre-computation of regridding weights and to enable re-use of regridding information for multiple fields.

Priority: 1.
Source: Required by all.
Status: Proposed.
Verification: Code inspection
Notes:

RG2.14 Dependence of field

For regridding schemes which might require field information, the required field information can be passed as arguments. Some higher-order regridding schemes require information on the gradient or other moments of a field. In such cases, this supplemental field information must be computed by the component and passed to the regridding function so that the regridding does not require detailed knowledge of operators or grid topology on every supported grid or field.

Priority: 1.
Source: Required by all.

Status: Proposed.

Verification: Code inspection

Notes: This requirement could also be satisfied by a later requirement for user-supplied regridding routines.

RG3 Regridding algorithms

This section contains requirements on regridding algorithms themselves.

RG3.1 Conservation

Regridding methods must be supplied which are conservative. Where possible, higher-order conservative methods should also be supplied. This requirement applies only to ESMF grids which have an area (2-d), volume (3-d) or linear region (1-d) associated with them such that conservation is well defined.

Priority: 1.

Source: POP,CICE,CAM required, NSIPP, CCSM-CPL, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Methods exist for both first and second-order conservative schemes in 1-d and 2-d [5]. Conservative methods for 3-d field (eg Monte Carlo or 3-d extensions to the above methods) are more difficult and may have a lower priority. High-order conservative schemes are more expensive and no schemes higher than second-order have been implemented.

MI - Some grids may have a higher-order integration method associated with them (overlapping functions as weights), potentially making conservative regridding difficult and expensive.

RG3.1.1 Verification of conservation

A method for verifying conservation must be supplied.

Priority: 2.

Source: CCSM-CPL.

Status: Proposed.

Verification: Unit test.

Notes: For error checking and testing.

RG3.2 Monotonicity

Monotone regridding methods must be supplied.

Priority: 2.

Source: CAM-FV, CAM-EUL, MIT.

Status: Proposed.

Verification: Unit test.

Notes: CAM-FV vertical remapping requires this. Biogeochemical models may need this. First-order conservative schemes are generally monotone by construction, so this could be satisfied by the conservation requirement for gridded data. 1-d monotone schemes are required for some hybrid and Lagrangian vertical coordinate schemes.

RG3.3 Higher-order schemes

Regridding methods which are higher than first order must be supplied. This is required for preventing “patchwork” patterns when regridding from coarse to fine grids and for preventing discontinuities in gradients of regridded fields.

Priority: 1.

Source: POP, CICE, CAM required, CCSM-CPL, NCEP-GSM, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes: This will require either internal approximations to gradients (eg bilinear, bicubic, trilinear) or will require the user to pass gradient information (eg second-order conservative methods). See requirements on field dependence. Also, this requirement can be in conflict with monotonicity requirements. Options to specify what happens to higher-order at boundaries are needed.

RG3.4 Vector fields in physical space

Regridding methods must be available for regridding a horizontal vector field with components aligned with physical directions (eg zonal-meridional or x-y), where the physical direction may be inferred by the grid type or specified by user.

Priority: 1.

Source: POP, CICE, CAM required, NSIPP, CCSM-CPL, NCEP-SSI, MIT.

Status: Proposed.

Verification: Unit test.

Notes: In spherical coordinates, meridional velocity components may be improperly handled except in simple latitude-longitude grid combinations.

RG3.5 Vector fields in logical space

Regridding methods must be available for regridding a horizontal vector field with components aligned along grid logical directions. Logical directions here refer to directions parallel and perpendicular to cell sides. Such a method would correctly handle flow through coordinate singularities such as the poles in spherical coordinates.

Priority: 2.

Source: POP, CICE (CCSM) desired, NSIPP, CCSM-CPL (desired), MIT.

Status: Proposed.

Verification: Unit test.

Notes: Currently working on whether it is even possible to do this in all cases, but willing to make the attempt. Conversion to 3-d Cartesian components during the remapping is one option.

RG3.6 Regridding based on index space

Methods must be available for regridding based only on logical indices of grid points and thus only on distributed grid information. Such a function is useful for nested grid and multi-grid applications where no physical grid information is required for creating the regridding.

Priority: 1.

Source: WRF required, NSIPP, MIT.

Status: Proposed.

Verification: Unit test.

Notes: Will need a general way to specify stencils

RG3.6.1 Index space changes

A method must be supplied for rapidly changing the regridding in cases where indices of one grid shift in relation to the other grid (eg as a nested grid moves in relation to its parent). The regridding in this case would be utilizing the same stencil and weights; only the addresses of the grid points would shift. Because of the simple nature of this operation, this requirement provides an efficient short-cut, avoiding re-creating a regridding using calls to create or destroy methods.

Priority: 1.

Source: WRF required.

Status: Proposed.

Verification: Unit test.

Notes: A requirement for this exists for the distributed grid, so Regrid would utilize the distributed grid functionality to accomplish this.

RG3.7 Fourier transforms

Methods shall be supplied for regridding between physical space and Fourier space. The adjoints shall also be supplied. Ordering in Fourier space will be defined by the distributed grid. This requirement applies only to grids consistent with the Fourier transform (eg lat/lon grids, reduced grids, spectral elements, etc.).

Priority: 1.

Source: NCEP-GSM, NCEP-SSI (milestone), NSIPP.

Status: Proposed.

Verification: Unit test.

Notes:

RG3.7.1 Return types for Fourier modes

Results of Fourier transforms can be returned as either complex numbers or as real numbers in a specified order.

Priority: 1.

Source: NCEP-GSM, NCEP-SSI.

Status: Proposed.

Verification: Unit test.

Notes:

RG3.7.2 Parallel implementations

Distributed FFT algorithms will be supported for a limited number of specific configurations. Note that a transpose algorithm (in which a local serial transform is combined with data transposes to redistribute data) will always be supported for the general case.

Priority: 1.

Source:

Status: Proposed.

Verification: Unit test.

Notes: Distributed algorithms make assumptions about the placement of both input and output data and support for all possibilities may be prohibitive.

RG3.8 Legendre transforms

Methods shall be supplied for regridding between spectral space and Fourier space. The adjoints shall also be supplied. This requirement applies only to grids consistent with the Legendre transform (the data must be located at appropriate quadrature points). Both scalar and vector fields must be supported.

Priority: 1.

Source: NCEP-GSM, NCEP-SSI (milestone), NSIPP.

Status: Proposed.

Verification: Unit test.

Notes:

RG3.8.1 Data types for Fourier modes

Legendre transforms must support Fourier modes stored as either complex numbers or as real numbers in a specified order.

Priority: 1.

Source: NCEP-GSM, NCEP-SSI.

Status: Proposed.

Verification: Unit test.

Notes: Companion to the Fourier requirement above.

RG3.8.2 Parallel implementations

Distributed Legendre algorithms will be supported for a limited number of specific configurations. Note that a transpose algorithm (in which a local serial transform is combined with data transposes to redistribute data) will always be supported for the general case.

Priority: 1.

Source:

Status: Proposed.

Verification: Unit test.

Notes: Similar to the Fourier requirement, distributed algorithms make assumptions about the placement of both input and output data and support for all possibilities may be prohibitive.

RG3.9 Other functional transforms

Methods shall be supplied for regridding using user-supplied matrices, particularly between functional space and physical space. The adjoints shall also be supplied. The grids again must be consistent with the functional transform being applied.

Priority: 1.

Source: NCEP-SSI (milestone), NSIPP, MIT.

Status: Proposed.

Verification: Unit test.

Notes: MI - The NCEP-SSI transforms between vertical EOF space and vertical model levels.

RG3.10 Interpolating from gridded data to ungridded data

All methods shall work for regridding FROM gridded data TO ungridded data, except that no conservation properties are required. Adjoints shall be supplied for interpolation TO ungridded data.

Priority: 1.

Source: NCEP-SSI (milestone), PSAS (not milestone), NSIPP, MIT.

Status: Proposed.

Verification: Unit test.

Notes:

RG3.11 Interpolating from ungridded data to gridded data

Methods for regridding FROM ungridded data TO gridded data may be supplied (eg nearest-neighbor distance-weighted schemes).

Priority: 3.

Source: MIT.

Status: Proposed.

Verification: Unit test.

Notes: Generally, operations like this will be covered by data assimilation schemes, but simple methods may be useful for other model-data comparisons.

RG3.12 User-supplied regridding methods

It shall be possible for users to supply their own regridding routines. This is especially useful for regriddings that are strongly dependent on model fields.

Priority: 2.

Source: CAM-FV, NSIPP, CCSM-CPL, MIT.

Status: Proposed.

Verification: Unit test.

Notes: The implementation report will examine use of function pointers or their equivalent, but implementation may run into other interface issues.

RG4 Other utilities

The following are utilities related to regridding which should be made public.

RG4.1 Exchange grid

A method for constructing a new grid formed by the intersecting cells of two grids shall be available.

Priority:

Source: NSIPP, MIT.

Status: Proposed.

Verification: Unit test.

Notes:

Part IX

Infrastructure Utilities: Time Management

1 Authors, target codes and review team

Authors: Chris Hill, Brian Eaton, Cecelia DeLuca

Review Date: 21 March, 2002

Target Codes

GFDL-SPEC, BGRID, MOM4 (GFDL)

HIM

CAM-EUL

FVCAM, PSAS

POP, CICE

WRF

MITgcm

NSIPP-ATM and NSIPP-OCN (NSIPP)

NCEP-ATM, SSI (NCEP)

Other Reviewers:

Reviewers

Balaji

Hallberg

Eaton

Sawyer

Jones

Michalakes

Hill

Suarez and Keppenne

Iredell

DeLuca, Neckels

2 Time management background

Many of the components that will run and interact within the ESMF are prognostic simulation and data assimilation codes employing time-stepping approaches to solve a numerical implementation of a set of mathematical equations. Coordinating component to component interactions and coordinating interactions between components and external systems, from which data is ingested or to which data is exported, requires precise notions of time. The role of the ESMF Time Manager (TimeMgr) is to provide a standard set of "time services" to components running under ESMF. The ESMF time management concepts have many parallels with real alarm clocks and with modern electronic scheduling systems. However, ESMF applications have a number of special requirements that go beyond the facilities that have become standard in mainstream software. The TimeMgr is designed to meet both the conventional needs and specialized demands of Earth system applications.

2.1 Location

The time management set of "time services" will be part of the ESMF Infrastructure layer. Including these services within the ESMF Infrastructure layer permits

- development of multiple components with compatible notions of time;
- development of a robust, core library of common time functions;
- subsequent community development of common higher-level time operations.

2.2 Scope

The time manager is not intended to be a comprehensive set of services for all time related operations. For example, it will not include a complete time-zone capability that supports translation between all of Earth's more than 300 timezones. Nor will it include algorithms that calculate detailed orbital quantities such as perihelion, obliquity and precession. However, the time manager will provide a generic foundation for developing libraries that do provide such

custom, specialized time services. It will support this in a way that permits easy interoperability and code sharing. As such, the time manager is anticipated to be a library of software that is targeted at both ESMF component developers and at specialized library developers.

3 Time management summary of requirements

The basic capabilities required by the time manager are satisfied by the concepts of time intervals, time instants, clocks and alarms, as defined in Section ??.

Time intervals and time instants are the computational building blocks of the TimeMgr library. Time intervals, which are time periods independent of any calendar, support operations such as add, subtract, compare size, reset value, copy value, and subdivide by a scalar. Time instants, which are moments in time associated with specific calendars, can be incremented or decremented by time intervals, compared to see which of two time instants is later, differenced to obtain the interval between two time instants, copied, reset, and manipulated in other useful ways. Time instants support a host of different queries, both for values of individual time instant components such as year, month, day and second, and for derived values such as day of year, middle of current month and Julian day. It is also possible to retrieve the value of the hardware realtime clock in the form of a time instant.

Since climate modeling, numerical weather prediction and other Earth system applications have widely varying time scales and require different sorts of calendars, the TimeMgr must provide a wide range of time specifiers, spanning nanoseconds to years. The set of supported calendars includes Gregorian, no-leap, Julian, and 360-day. The TimeMgr also supports a user-specified calendar.

Although it is possible to repeatedly step a time instant forward by a time interval using arithmetic on these basic types, it is useful to identify a higher-level concept that encapsulates this function. We refer to this capability as a clock, and include in its required features the ability to store reference times such as the start and stop time instants of a model run, to check when time advancement should cease, and to query the value of quantities such as the previous and current time instants. The TimeMgr must include methods that return a flag value when a periodic or unique event has taken place; we refer to these as alarms. Applications may require temporary or multiple clocks and alarms.

For operations on time types, finite precision arithmetic that has defined semantics for both floating point and integer operands is required. Arithmetic based on rational fractions, with support for arbitrarily accurate drift-free (i.e. *exact*) clocks, is desired to extend the capabilities of target applications.

The time manager must satisfy the framework-wide requirements for the ESMF described in the *ESMF General Requirements* (see [12]), including requirements for supported platforms, robust error handling and real and integer precision.

4 Time management abbreviations

Table 1: Specifiers for time intervals, time instants, and calendar intervals.

Name	Meaning
YR	Integer year.
MM	Integer month of year.
D	Integer number of days.
d	Floating point number of days.
H	Integer number of hours.
h	Floating point number of hours.
M	Integer number of minutes.
m	Floating point number of minutes.
S	Integer number of seconds. May need <i>nd</i> form.
s	Floating point number of seconds.
MS	Integer number of milliseconds.
ms	Floating point number of milliseconds.
TS	Integer number of 1/10,000 seconds.
US	Integer number of microseconds.
NS	Integer number of nanoseconds.
DD	Day of month.
O	Time zone offset in integer number of hours and minutes.
_nd	Suffix to indicate integer + $\frac{n}{d}$ form, where <i>n</i> and <i>d</i> are integers. For example, S_nd has an integer second component and a fractional second component. _nd provides a mechanism for supporting exact behavior.

5 Time management requirements

TMG1 Time intervals

TMG1.1 Specifying time intervals

Time intervals specified by the user shall be represented at the interface by the sum of some subset of the quantities **D**, **S** and **S_nd**, **MS**, **NS**, **d**, **m**, **s**, **ms**, **ns** together with an optional sign. Shortcut interfaces that provide for the following interval representations: **S**; **D+S**; **s** are required.

Priority: 1.

Source: WRF uses **S** to specify timesteps and will use **S_nd**;

CAM-EUL, CAM-FV, FMS codes use a **D+S** representation for time intervals; NSIPP, HIM and MITgcm use **s**; POP, CICE require all but **S_nd**, **NS**, **m**, **ms**; NCEP requires signed **s**, **h** and **D+H+M+S+MS**; WRF and GFDL desire **S_nd** for model enhancement

Status: Approved-1 for all except **S_nd**; approved-2 for **S_nd**.

Verification: Interface inspection, unit test.

Notes: Methods that require time interval specification can be designed with a generic interface. Such an interface would allow the user to select the subset of time specifiers included in the argument list. This would satisfy the needs of target codes and would also allow the interface to be extended to include other time units. The generic interface would be implemented in F90 using named optional arguments. The same approach is suggested for time instants.

TMG1.2 Time intervals as return values

It shall be possible to return the value of a time interval in a variety of representations. These shall include the combinations listed in ?? plus **d, h, m, s, ms**. For returned time periods, the value of each time quantity, such as **S**, is bounded by the next larger quantity in the representation. For example, for a **D+S** representation under a Gregorian calendar, the value **S** that is returned will be such that $S \leq 86399$ (the length of a Gregorian calendar day in seconds).

Priority: 1.

Source: CAM-EUL requires **D+S** and **d**; GFDL requires **D+S** and **S**; NCEP requires **d, h, m, s, ms**; NSIPP, HIM, MITgcm require **s**; POP, CICE require **d, s**.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: A general conversion method between time representations may be a compact way to meet this requirement.

TMG1.3 Resolution

A clear statement and consistent implementation of the resolution used in time interval calculations is required. A default precision of at least **US** is required to meet the current needs of target applications; **NS** is desired by some applications for extensibility.

Priority: 1.

Source: CAM-EUL, GFDL, CAM-FV require **S**; NCEP-ATM, SSI, NSIPP require **MS**; WRF requires **TTS**; MITgcm requires **US**; POP, CICE require **US**; MITgcm desires **NS**.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: It should be possible to accommodate the highest resolution desired, **NS**, without difficulty.

TMG1.4 Range of time intervals

It shall be possible to use the TimeMgr to run an application over a range of at least 200,000 years.

Priority: 1.

Source: CAM-FV, NCEP, POP, CICE require at least 20,000 years; CAM-EUL, GFDL, MITgcm and NSIPP require a range of at least 200,000 years.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: should be possible to encompass the largest range desired. A pair of 64-bit integers can represent, in seconds and atto-seconds ($10^{-18}s$), a time range of about 300 billion years!

TMG1.5 Operations

TMG1.5.1 Change value

A time interval may have its value changed.

Priority: 1.

Source: Required for GFDL, MITgcm, CAM-FV, NSIPP, CAM-EUL, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG1.5.2 Copy

A time interval may be copied to another time interval.

Priority: 2.

Source: Required for GFDL, NSIPP, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: This can be satisfied by using a time interval query TMG1.2 and change value combination.

TMG1.5.3 Comparison

A pair of time intervals can be compared for magnitude, equality, inequality and ordering.

Priority: 2.

Source: Required for GFDL, NSIPP, CAM-EUL, CAM-FV, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG1.5.4 Increment and decrement

A time interval can be incremented or decremented by another time interval.

Priority: 1.

Source: Required for GFDL, NSIPP, NCEP-ATM, CAM-EUL, CAM-FV, POP, CICE, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG1.5.5 Division

A time interval can be divided by another time interval, resulting in a fraction that expresses the relative magnitudes of the two time intervals.

Priority: 2.

Source: Required for GFDL codes.

Status: Approved-2.

Verification: Interface inspection, unit test.

Notes: This is useful for some time interpolation procedures. Supporting exact division implies supporting an `_nd` fraction representation.

TMG1.5.6 Subdivision

A time interval may be divided into an integer number of equally sized intervals.

Priority: 2.

Source: Required for CICE, GFDL; desired for POP and CAM-FV, WRF.

Status: Approved-2.

Verification: Interface inspection, unit test.

Notes: Exact subdivision requires `S_nd` support and possibly an `_nd` representation of the divisor. This operation can be done by application code provided the time interval query capability `??` returns a full description.

TMG1.5.7 Multiplication

A time interval may be multiplied by an integer or a floating point number.

Priority: 2.

Source: Required for GFDL, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: Can be done by application code provided query capability ?? returns a full description. Exact multiplication requires `_nd` support for multiplication factor.

TMG1.5.8 Magnitude

The absolute value of a time interval is required.

Priority: 2.

Source: Desired for GFDL codes (FMS time types are positive).

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: Can be done by application code provided query capability TMG1.2 returns a full description. Exact magnitude requires `Snd` support.

TMG1.5.9 Return in string format

Time intervals can be returned formatted as strings.

Priority: 3.

Source: Required for NCEP; desired by POP, CICE, MITgcm, WRF.

Status: Approved-2.

Verification: Interface inspection, unit test.

Notes: Formatting could be done by applications, assuming a full query capability. Discussion of the format flexibility that might be required can be found under [10].

TMG2 Time instants

TMG2.1 Units and representation

Time instants specified by the user shall be represented at the interface by the combination of some subset of the quantities **YR, MM, DD, H, M, S, MS, NS, O, d,h,m,s** and the calendar type. Shortcut interfaces shall be provided for the following time instant representations **s; YR+MM+DD+S, YR+MM+DD+HH+M+S**. A generic interface for a full specification of all terms shall also be provided.

Priority: 1.

Source: NSIPP, HIM, MITgcm require **s**; CAM-EUL and CAM-FV require **YR+MM+DD+S**; MITgcm, GFDL requires **YR+MM+DD+HH+M+S**; POP, CICE require **YR+MM+DD+HH+M+S, d, s**;

WRF requires **YR+MM+DD+H+M+S, YR+MM+DD+H+M+S_nd**

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: The ranges of the date components **YR, MM, DD** are calendar specific. The ranges of the time of day components **M, s** are not constrained to be between 0 and 60. It is, for example, possible to express the time of day using the **s** component only.

TMG2.2 Consistency with time interval

Time resolution of a time instant must be consistent with that of a time interval. Consistency is defined in the following way: the result of incrementing/decrementing any representable time instant by any representable time interval must be a representable time instant, assuming it lies within the valid range of the calendar.

Priority: 1.

Source: All applications require this basic constraint.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.3 Supported calendars

TMG2.3.1 Gregorian calendar

Time instants may use the Gregorian calendar and UTC times. The required range of dates is as wide as possible, but certainly not less than 10,000 years.

Priority: 1.

Source: Required for NSIPP, CAM-FV, CAM-EUL, POP, CICE, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: The available range of Gregorian dates using the standard Fliegel et al.[9] algorithm is from November 25, -4713 onward. Support for leap seconds is **not** required.

TMG2.3.2 No-leap calendar

Time instants may use a no-leap year calendar which is the same as the Gregorian except that it does not include leap year corrections.

Priority: 2.

Source: Required for CAM-EUL, GFDL, POP, CICE, MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.3.3 Julian calendar

Full Julian calendar support is required.

Priority: 2.

Source: Required for GFDL, NSIPP.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.3.4 360-day and generic calendar

The TimeMgr should provide support for a generic calendar.

Priority: 1.

Source: Required for GFDL, POP, CICE, MITgcm, desired for NSIPP.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: Often it is useful to perform an idealized simulation with a calendar that is an approximation to an actual calendar. A commonly used configuration is a 360 day year containing 12 months of 30 days each. Each day is

exactly 86400 seconds. The period of rotation is assumed to be exactly one day and the orbital period around the Sun is assumed to be exactly one year. These settings are a useful approximation for the Earth. However, for other planets, or for idealized parameter space exploration experiments, the year length and day length need to be adjusted. A generic calendar may be specified by the number of days in each month of the year or by the length of the year, and the day length. The year length need not be an integer number of days, for example on Venus a year would be 0.926 days. The range of the year number for generic calendars is -200,000 to 200,000.

TMG2.3.5 No calendar option

Time instants may use a “no calendar” option.

Priority: 3.

Source: May be useful for CAM-EUL and MITgcm, WRF.

Status: Approved-2.

Verification: Interface inspection, unit test.

Notes: It is possible to achieve this capability by ignoring the year in TMG2.3.4. The **YR** and **MM** components of a time instant are ignored under the “no calendar” option. The range of the day number, **DD**, will be at least the number of days in 200,000 years.

TMG2.4 Operations

TMG2.4.1 Change time instant value

All the components, except the calendar, of a time instant can be changed.

Priority: 1.

Source: Required by CAM-EUL, GFDL, NSIPP, CAM-FV, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.4.2 Copy

A time instant can be copied to another time instant.

Priority: 2.

Source: Required for GFDL, NSIPP, MITgcm, desired for CAM-EUL, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.4.3 Comparison

A pair of time instants can be checked for equality or to determine which is the later or earlier of the pair.

Priority: 1.

Source: Required for GFDL, POP, CICE, CAM-EUL, CAM-FV, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.4.4 Increment or decrement by time interval

Time instants can be incremented or decremented by time intervals.

Priority: 1.

Source: Required for GFDL, POP, CICE, CAM-FV, CAM-EUL, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: These methods are *exact*.

TMG2.4.5 Increment or decrement by a calendar interval

Time instants can be incremented or decremented by years, months, or seconds.

Priority: 2.

Source: Desired for CAM-EUL, NSIPP, MITgcm.

Status: Approved-2.

Verification: Interface inspection, unit test.

Notes: Clocks conventionally operate so that increments by a month are rounded down to nearest date within the month you incremented up to! e.g. Jan 31 + **MM** == Jan 30th + **MM** == Jan 29th + **MM** == Feb 28th (or 29th in a leap year). Jan 31 + **2MM** == March 31. Support for **S_{nd}** will be required.

TMG2.4.6 Interval between time instants

A method shall be provided to calculate the time interval between a pair of time instants.

Priority: 1.

Source: Required for CAM-FV, GFDL, POP, CICE, CAM-EUL, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: These methods are *exact*.

TMG2.4.7 Return in string format

Time instants may be returned formatted as strings.

Priority: 3.

Source: Desired for NCEP, POP, CICE, CAM-EUL, MITgcm, WRF.

Status: Approved-2.

Verification: Interface inspection, unit test.

Notes: Formatting can be done by applications.

TMG2.5 Queries

TMG2.5.1 Standard queries

A time instant may be queried for any of the values of **YR, MM, DD, H, M, S, O, MS, NS, d, h, m, s**, and the calendar type. A query make take an **O** value which specifies an amount by which the returned time-instant is offset relative to its internal **O=0** value.

Priority: 1.

Source: All applications require a query that supports some subset of the list above; CAM-EUL requires **YR, MM, DD, S**.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: Time instant components will be returned in canonical form with **H** in the range 0 to 23, **M** in the range 0 to 59, and **s** in the range 0 to 59.999... depending on precision.

TMG2.5.2 Query day of year

A time instant may be queried for the day of the year represented as a floating point number with the fractional part representing the time of day.

Priority: 2.

Source: Required for NCEP, CAM-EUL; desired for NSIPP, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.5.3 Query day of week

A time instant may be queried for the day of the week.

Priority: 2.

Source: Required for NCEP, desired for NSIPP.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.5.4 Query day of month

A time instant may be queried for the day of the month.

Priority: 3.

Source: Desired for CAM-EUL.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.5.5 Query middle of month

A time instant may be queried for the time instant of the middle of the month that the time instant falls in.

Priority: 2.

Source: Required for POP, CICE, NSIPP.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG2.5.6 Query julian day

A time instant may be queried for its Julian day.

Priority: 1.

Source: Required for NCEP.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: NCEP codes rely on this.

TMG2.5.7 Query hardware realtime clock

Return the actual hardware realtime clock time instant in the UTC time zone.

Priority: 2.

Source: Required for NCEP, CAM-EUL; desired for NSIPP, MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: In order to support synchronization with an external clock, the hardware realtime clock must be returned in a format valid as an input to a time instant change value method.

TMG3 Clocks

TMG3.1 Clock initialization

A clock is initialized by start and stop time instants, timestep interval, and an optional reference time instant. The default value of the reference time instant is the start time instant. The reference time is used by the clock to provide time coordinate values for the simulation in the form of elapsed time since a reference time.

Priority: 1.

Source: Required for CAM-FV, CAM-EUL, MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG3.2 Multiple clocks

Components should be able to create and manipulate multiple clocks.

Priority: 1.

Source: Required for MITgcm, WRF, desired for CAM-FV.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: In many numerical approaches several different time-steps are used for different elements of the system. Multiple clocks are important to keep track of these different time-steps. Ensemble simulations may also require members to proceed with different temporal trajectories, all within a single component.

TMG3.3 List of clocks

Components should be able to get a list of their clocks.

Priority: 2.

Source: Required for NSIPP, desired for CAM-FV, MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG3.4 Operations

TMG3.4.1 Advance method

A clock has an advance method. When the clock is advanced the clock's time instant is incremented by the clock's current timestep interval.

Priority: 1.

Source: Required for CAM-FV, NSIPP, CAM-EUL, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: This functionality can easily be achieved without the concept of a clock, by incrementing a generic time instant with a time interval. The clock concept bundles this capability with others that are closely related, such as the ability to get the value of the previous time instant.

TMG3.4.2 Reset timestep interval

A clock's timestep interval can be changed.

Priority: 1.
Source: Required for CAM-FV, POP, MITgcm, WRF.
Status: Approved-1.
Verification: Interface inspection, unit test.

TMG3.4.3 Change clock current time instant

A clock's current time instant can be changed. This action causes the previous time instant to be reset to the current time instant. The clock timestep counter does not change.

Priority: 2.
Source: Required for CAM-EUL, NSIPP, MITgcm.
Status: Approved-1.
Verification: Interface inspection, unit test.

TMG3.4.4 Restore clock state

A clock can be returned to its exact state from a previous run. This is to support a component's restart capability. The clock must be able to provide its state to the component, and be able to reset its state. The component is responsible for the persistence of the clock's state data.

Priority: 1.
Source: Required for CAM-FV, POP, CICE, CAM-EUL, MITgcm, WRF.
Status: Approved-1.
Verification: Unit test.

TMG3.4.5 Synchronize with external clock

The ability to synchronize or "attach" a clock to an external source should be supported.

Priority: 2.
Source: Required for NCEP.
Status: Approved-1.
Verification: ? **Notes:** Forecast scenarios require latching key events to actual wall-clock time.

TMG3.5 Queries

TMG3.5.1 Query number of timesteps

A clock can be queried for the number of times the advance method has been called. This is also known as the timestep number. It is initialized to zero and increases monotonically.

Priority: 2.
Source: Required for POP, CICE; desired for NSIPP, CAM-EUL, MITgcm, WRF.
Status: Approved-1.
Verification: Interface inspection, unit test.

TMG3.5.2 Query timestep interval

A clock can be queried for the current timestep interval.

Priority: 1.
Source: Required for POP, CAM-FV, CAM-EUL, MITgcm, WRF.
Status: Proposed-1.
Verification: Interface inspection, unit test.

TMG3.5.3 Query start, stop, reference time

A clock can be queried for the start, stop, or reference time instants.

Priority: 1.

Source: Required for POP, CICE, CAM-FV, CAM-EUL, MITgcm, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG3.5.4 Query current or previous time instants

A clock can be queried for the previous or current time instants. The current time instant is equal to the start time instant until the first call of the advance method. The previous time instant is equal to the current time instant until the first call of the advance method.

Priority: 2.

Source: Required for POP, CICE, CAM-EUL; query current time instant required by CAM-FV, MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG3.5.5 Query current or previous simulation times

A clock can be queried for the previous or current simulation times. The current simulation time is the time interval between the current time instant and the reference time instant. Previous simulation time is defined analogously.

Priority: 2.

Source: Required for POP, CICE; desired for CAM-FV; query current simulation time required by MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG3.5.6 “Is Later” query

A clock can be queried for whether or not the current time instant equals or is later than the stop time instant.

Priority: 1.

Source: Required for POP, CICE, CAM-FV, MITgcm; desired for CAM-EUL, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: This capability may be necessary in order to implement alarms.

TMG4 Alarms

TMG4.1 Alarm initialization

An alarm is initialized by specifying the clock to which it is associated and its ringing times.

Priority: 1.

Source: Required for CAM-EUL, CAM-FV, NSIPP, POP, CICE, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG4.2 Multiple alarms per component

Components should be able to create and manipulate multiple alarms. Each alarm is associated with one and only one clock; one clock may be associated with multiple alarms.

Priority: 1.

Source: Required for CAM-EUL, CAM-FV, NSIPP, POP, CICE, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG4.3 List and print of alarms

Components should be able to get a list and create a print record of their alarms.

Priority: 2.

Source: Required for NSIPP.

Status: Approved-2.

Verification: Interface inspection, unit test.

TMG4.4 Alarm states

An alarm can be either on (i.e., ringing) or off (i.e., quiet). The alarm on/off state is set when the alarm is initialized, and can be queried at any time by the application. The alarm is turned on if the clock's current time instant is later than or equal to the current ringing time and the previous time instant was before the current ringing time. Otherwise the alarm is off.

Priority: 1.

Source: Required for CAM-EUL, CAM-FV, NSIPP, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG4.5 Ring criteria

TMG4.5.1 Ring at time instant

An alarm can be set to ring at a single time instant.

Priority: 1.

Source: Required for NSIPP, POP, CICE, CAM-EUL, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: This capability can also be accomplished with a comparison of time instants.

TMG4.5.2 Ring at interval

An alarm can be set to ring at regular intervals, for example at the beginning, middle or end of some period. The ringing times are specified by a starting time instant (offset), a time interval, and an optional stopping time instant.

Priority:

Source: Required for NSIPP, CAM-FV, POP, CICE, CAM-EUL.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG4.5.3 Initial ring state

At initialization it should be possible to specify whether ringing is enabled immediately, or whether ringing tests are deferred until after the next clock update.

Priority: 2.

Source: Required for NSIPP, POP, CICE.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG4.6 Alarm turn-off

Alarms are only turned on by the clock. A ringing alarm may be turned off either by a clock update operation that assesses whether the current time instant lies within an interval during which the alarm should be on, and if not turns it off; or an operation that simply turns the alarm off. application.

Priority: 1.

Source: Required for NSIPP, CAM-FV, POP, CICE, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG4.7 Restore alarm state

An alarm can be returned to its exact state from a previous run. This is to support a component's restart capability. The alarm must be able to provide its state to the component, and be able to reset its state. The component is responsible for the persistence of the alarms state data.

Priority:

Source: Required for NSIPP, POP, CICE, CAM-FV, CAM-EUL, WRF.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG4.8 Alarm queries

Alarms have query methods for previous, current, and next ringing times. When an alarm is initialized the previous ringing time is set to the current ringing time.

Priority:

Source: Required for POP, CICE, desired for NSIPP.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG5 Accuracy of calculations

Accuracy of calculations with time instants and time intervals.

TMG5.1 Exact increment and decrement

Incrementing or decrementing time instants by exact time intervals is exact. By exact incrementing of time instants by time intervals we mean the following. Given a time instant τ_{init} and a time interval i_1 , incrementing τ_{init} by i_1 N successive times gives the same result as incrementing τ_{init} by an interval i_N where i_N is the result of incrementing i_1 by i_1 , $(N - 1)$ times. Exact decrementing is defined analogously.

Priority: 1.

Source: Required for CAM-FV, GFDL, NSIPP, CAM-EUL, desired for MITgcm, WRF.

Status: Approved-1.

Verification: Unit test.

TMG5.2 Exact interval calculation

Calculating the interval between exact time instants is exact.

Priority:

Source: Required for CAM-FV, GFDL, NSIPP, CAM-EUL, desired for MITgcm, WRF.

Status: Approved-1.

Verification: Unit test.

Notes: Given time instants τ_{init} and τ_{final} , then the calculated difference between these instants, i_1 , has the property that incrementing τ_{init} by i_1 gives τ_{final} .

TMG5.3 Exact subdivision

Exact division of time intervals into an integral number of subintervals.

To ensure a drift free clock the following behavior must be true:

if adding (or subtracting) a time interval, i_1 , to a time instant τ_{init} yields a new time instant τ_{final} i.e.

$$\tau_{final} = \tau_{init} + i_1$$

then it must be possible to specify a fractional time interval $i_2 = \frac{1}{2} \times i_1$ such that

$$\tau_{final} = \tau_{init} + i_1 \equiv \tau_{init} + i_2 + i_2$$

and similarly for $i_3 = \frac{1}{3} \times i_1$, $i_{n,m} = \frac{n}{m} \times i_1$.

Priority:

Source: Desired for CAM-FV, GFDL, MITgcm, WRF.

Status: Approved-2.

Verification: Unit test.

Notes: Such behavior is not possible with finite-precision floating point arithmetic. The fractional second part of both time instants and time intervals must be able to exactly represent rational values. If partial seconds represented as rational numbers is supported, then the precision of time intervals and time instances must be $1/N$ seconds where N is the largest value of the denominator. Thus an **_nd** representation is required to fully support this.

TMG5.4 Floating point accuracy consistent with time step

Clock accuracy comparable with numerical scheme accuracy should be supported. Applications that employ an adaptive time-step for their numerical procedures need to be able to drive their time manager clocks with that step. This should be possible even for application time steps that are arbitrary floating point numbers i.e. not a whole number of seconds, or minutes.

Priority: 1.

Source: Required for NSIPP, MITgcm.

Status: Approved-2.

Verification: Unit test.

TMG6 Cross-component clock and alarm queries

TMG6.1 Cross-component query

Components should be able to query the clock(s) or alarms of another component and components should be able to manipulate certain clock(s) and alarms of another component.

Priority:

Source: Desired for POP, CICE, NSIPP, MITgcm.

Status: Approved-2.

Verification: Unit test.

Notes: Synchronized shutdown notifications will be sent to components through clock and alarm settings. These notifications may be generated by a high-level control program or a coupler component. However, components should be able to determine which clocks and alarms can be externally manipulated so that they can assume that certain clocks and alarms are private.

TMG6.2 Clock and alarm labels

Components should be able to "label" their clocks and alarms.

Priority:

Source: Required for POP, CICE, MITgcm; desired for NSIPP.

Status: Approved-2.

Verification: Unit test.

Notes: When coordinating notions of time among components with several clocks or alarms, it is important to be able to identify the role of each clock or alarm. For instance, a component might want to declare a master clock that defines the overall time within a component, or components might need to declare a standard alarm that can be used by a higher-level driver layer or a coupler component to trigger the generation of restart information or to trigger a clean shutdown.

TMG7 General computational requirements

TMG7.1 Error handling

The TimeMgr services must conform to the error handling specifications described in the *ESMF General Requirements* document [12] and, when available, the *ESMF Error Handling Requirements* document. The requirements below are some special error handling capabilities needed for the TimeMgr .

TMG7.1.1 Check validity

A time interval or time instant may have its validity checked. An invalid time interval should create a recoverable error condition so that a component can decide on the error action.

Priority: 1.

Source: Required for CAM-EUL, CAM-FV, GFDL, NSIPP, MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG7.2 Overloaded arithmetic operators

Overloaded arithmetic operator syntax should be used for time instant and time interval operations such as increment and decrement.

Priority: 2.

Source: Required for GFDL; useful for CAM-EUL.

Status: Approved-1.

Verification: Unit test.

TMG7.3 Automatic memory deallocation

It shall be possible to create a time interval, time instant, clock or alarm as a local variable, so that is not necessary to explicitly free the memory associated with it.

Priority: 2.

Source: Required for GFDL; desired for CAM-EUL, CAM-FV, NCEP, MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

Notes: This requirement implies that internal pointers are not allowed. If deletion is not automatic, a destructor will be required.

TMG7.4 Temporary objects

A shall be possible to create temporary clocks and alarms.

Priority: 2.

Source: Desired for NSIPP, MITgcm.

Status: Approved-1.

Verification: Interface inspection, unit test.

TMG7.5 Thread safety

Operations on time intervals and time instants will be thread-safe.

Priority: 3.

Source: Useful for CAM-EUL, MITgcm.

Status: Approved-1.

Verification: Unit test.

Reference Material

[1] *International Earth Rotation Service (IERS)* <http://hpiers.obspm.fr>

IERS Bulletins

IERS Constants

[2] *CSM Time Conventions*

[3] *OMG*

CORBA Time Service Specification Version 1.0 <ftp://ftp.omg.org/pub/docs/formal/00-06-26.pdf>

[4] US Naval Observatory

US Naval Observatory Time Pages

[5] W3C Date and Time Note

- [6] NIST Time and Date Material
- [7] A Calendar FAQ.
- [8] Some notes on the ISO8601 time and date specification standard.
- [9] Fliegel, H.F. and Van Flandern, T.C., Comm.ACM V11 N10, Oct 1968, p657 Press, W.H., et.al. (1986) 'Numerical Recipes', Cambridge, pp 10-13
- [10] The JAVA Calendar Class
- [11] Software Requirements. Karl Eugene Wiegers. Microsoft Press, 1999. ISBN 0-7356-0631-5.
- [12] y ESMF General Requirements. ESMF Joint Specification Team, 2002.

Part X

Infrastructure Utilities: Communication and Memory Kernels

1 Target Codes and Review Team

Sign-Off Date:	<Date>
Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CAM-EUL, CLM	Eaton
CAM-FV, PSAS	da Silva
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell
Other Reviewers:	DeLuca, Neckels

2 Background

The communication and memory kernels elements of ESMF provide a set of services that shield the upper levels of ESMF from system level functions. Two related areas are addressed. The basic communication primitives that are building blocks for both the data communication between and within components and the control messages that components may need to exchange.

One reason for isolating this section of ESMF is to help abstract the communication primitives to allow highly optimized code, targeted to specific platforms and to specific framework functions to be easily inserted into ESMF. The **CMK** layer will allow specialized forms of performance critical primitives to be employed without impacting application code or upper layers of ESMF code.

Channeling communication and ESMF memory allocation through a single layer is also designed to support detailed application-level monitoring.

Infiniband, LAPI example, IPC + LAPI, Myrinet global sum v. GM sum. Overlapping co-processor v. non-co-processor. overlap VIA, Gigaset, Servernet -> convergence FDDI

<Description of this class, module or utility set.>

2.1 Location

<Describe whether this software is in the infrastructure or superstructure, and other software modules that it interacts with.>

2.2 Scope

<Discussion of scope and restrictions.>

Table 2: <Table name>

Name	Meaning
IT1	<Item 1.>
IT2	<Item 2.>

2.3 Related Material

3 Communication and Memory Kernels Abbreviations

CMK1 Basic, portable MPI based transport

CMK1.1 A baseline MPI based build must be available

Priority: Priority 1

Source:

Status:

Verification:

Notes:

CMK2 Basic, portable threads based parallelism

Priority: Priority 1

Source:

Status:

Verification:

Notes:

Reference Material

- [1] Earth system modeling framework. <http://www.esmf.ucar.edu>, 2002.
- [2] A. Arakawa. Computational design for long-term numerical integration of the equations of atmospheric motion. *J. Comp. Phys.*, 1:119–143, 1966.
- [3] V. Balaji. Parallel numerical kernels for climate models. In ECMWF, editor, *ECMWF Teracomputing Workshop*. World Scientific Press, 2001.
- [4] R. Heikes and D. A. Randall. Numerical Integration of the Shallow-water Equations of a Twisted Icosahedral Grid. Part I: Basic Design and Results of Tests. *Monthly Weather Review*, 123:1862–1880, 1995.
- [5] P.W. Jones. First- and second-order conservative remapping schemes for grids in spherical coordinates. *Monthly Weath. Rev.*, 127:2204–2210, 1999.
- [6] D. Majewski, D. Liermann, P. Prohl, B. Ritter, M. Buchhold, T. Hanisch, G. Paul, and W. Wergen. The Operational Gblal Icosahedral-Hexagonal Gridpoint Model GME: Description and High-Resolution Tests. *Monthly Weather Review*, 130:319–338, 2002.
- [7] Ross J. Murray. Explicit generation of orthogonal grids for ocean models. *J. Comp. Phys.*, 126:251 – 273, 1996.
- [8] M. Rancic, R.J. Purser, and F. Mesinger. A global shallow-water model using an expanded spherical cube: Gnomonic versus conformal coordinates. *Quart. J. Roy. Meteor. Soc.*, 122:959 – 982, 1996.
- [9] Wiegers, K. E. *Software Requirements*. Microsoft Press, 1999.
- [10] Womack, B., Higgins, G. *Software Engineering Support of the Third Round of Scientific Grand Challenge Investigations. General Requirements Analysis*. NASA/CT Internal Document, 2002.

Part XI

Infrastructure Utilities: Configuration Attributes

1 Configuration Attributes Overview

2 Target Codes and Review Team

Sign-Off Date:	<Date>
Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CAM-EUL, CLM	Eaton
CAM-FV, PSAS	da Silva
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell
Other Reviewers:	DeLuca, Neckels

3 Introduction

4 Background

As part of their state Earth system models need to maintain ad-hoc sets of parameters. These parameters can be related to physical terms (for example mixing coefficients, length scales or time scales) or can be related to computational aspects (for example directory names, output and input locations). Some of these parameters will be explicitly set by user input at runtime, others may be automatically determined from system defaults or from other utilities. The parameters may be single valued or may be multi-dimensional. Parameters can be of various types float, integer, logical, string.

The configuration attributes element of ESMF will enable these parameters or **attributes** to be recorded and provided services for setting attributes from human readable text files and for saving attributes to persistent storage in appropriate formats.

Providing a standard service for configuration attributes will enable ESMF to provide enhanced interoperability capabilities. In particular the **attributes** facility will allow components to automatically share attribute settings.

4.1 Location

The configuration attributes element of ESMF will be part of the Infrastructure/Utilities area. Other upper layer tools may use the configuration attributes internally. User-level component code will also use the configuration attributes element.

4.2 Scope

Compile time and runtime setting of parameters will initially be through text based specification. Extensive automatic systems for choosing parameters or GUI based systems for manipulating parameters are not required within the initial ESMF development scope. Such systems are however highly desirable and it is envisaged that such systems would one-day interface with ESMF applications in part through the configuration attributes services. Another area that is outside the present scope of the initial ESMF project is automated, structured archival of attribute information along with other numerical experiment state. However, again this would be a useful concept to layer on top of ESMF.

Table 3: <Table name>

Name	Meaning
IT1	<Item 1.>
IT2	<Item 2.>

4.3 Related material

Property sheets used in many component based programming environments, for example the property sheet notion systems like Visual C++ (see the **CPropertySheet** and **CPropertyPage** MFC classes), .NET and J2SE (see **java.beans** class), have parallels with the concepts that need to be supported here. All the ESMF JMC codes contain services of this nature with widely varying degrees of sophistication (see for example MOM documentation, MITgcm documentation, CCSM documentation). In most cases the existing ESMF JMC codes use the Fortran NAMELIST facility. The widely used dot files and dot subdirectories used for all manner of application configuration on UNIX platforms also perform similar roles (see for example the **pine** mail reader documentation or the **OpenSSH** package documentation) to some of the functions envisaged here, as does the Windows registry concept (see **Windows registry** documentation).

5 Configuration Attributes Terms

NAMELIST An I/O feature supported by Fortran that defines a structured syntax for creating text files of initial variable settings and defines language features for compactly reading the files. Most ESMF codes use the NAMELIST features in Fortran, but the feature has several deficiencies and limited capabilities. The ESMF **configuration attributes** facility addresses these deficiencies and limitations. The syntax for NAMELIST files can be found in most Fortran manuals and tutorial texts.

<item2> glos:item2> <Description item 2.>

6 Configuration Attributes Abbreviations

7 Configuration Attributes Requirements

CA1 A human friendly format for parameter specification should be provided

CA1.1 Text files specifying parameters should allow comments

A simple form of # at the beginning of a comment line should suffice.

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

CA1.2 Text file parsing errors should at the least provide

file name, line of file, meaningful error message logged in an easy to find location

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

CA1.3 Text file syntax should be backwards compatible with NAMELIST formats.

The existing NAMELIST formats used by the JMC codes need to be parseable as Configuration Attributes. This could be through conversion or through a CA format that only extends the NAMELIST format i.e. that includes the NAMELIST format as a subset. The full CA format does not need to be compatible with the NAMELIST standard.

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

CA2 Attributes can be sub-classified

A system for grouping attributes into sub-categories within an ESMF application would be useful.

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

CA3 **Attributes can be optional and could be introduced at runtime**

There should be no need to specify all attributes. It should also be possible to introduce a new attribute e.g. *fred=7* without having to modify the attribute parser code. The Fortran NAMELIST requires that all attributes be listed in the parsing routine. This should not be mandatory in CA.

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

CA4 **A system for defaults and overrides should be supported**

It should be possible to have a set of default attribute settings and only require overrides to be specified.

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

CA5 **A system that is compatible with unique naming should be devised**

For example it should be possible to refer to an attribute with a name such as *gov.gfdl.mom4.timestepping.deltat*. It should be possible to determine whether the name matches uniquely or has multiple matches (e.g. for the above example an application with an ensemble of mom4 components would have multiple matches).

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

CA6 **Components should be able to query the attributes of other components**

This will be the main benefit of CA to ESMF.

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

CA7 Components should be able to set attributes to be writable by other components

Queries will be read-only, however it could also be useful to ultimately allow one component to set a value of another component's attributes. For example in ensemble simulations or computational steering a *driver* component could control other components.

Priority: <Priority 1-3>

Source: MITgcm

Status: <Proposed, Approved-1, Approved-2, Rejected, Implemented, Verified>

Verification: <e.g., Code inspection, Unit test, System test>

Notes: <Background, comments on design, implementation, etc.>

Reference Material

- [1] Earth system modeling framework. <http://www.esmf.ucar.edu>, 2002.
- [2] A. Arakawa. Computational design for long-term numerical integration of the equations of atmospheric motion. *J. Comp. Phys.*, 1:119–143, 1966.
- [3] V. Balaji. Parallel numerical kernels for climate models. In ECMWF, editor, *ECMWF Teracomputing Workshop*. World Scientific Press, 2001.
- [4] R. Heikes and D. A. Randall. Numerical Integration of the Shallow-water Equations of a Twisted Icosahedral Grid. Part I: Basic Design and Results of Tests. *Monthly Weather Review*, 123:1862–1880, 1995.
- [5] P.W. Jones. First- and second-order conservative remapping schemes for grids in spherical coordinates. *Monthly Weath. Rev.*, 127:2204–2210, 1999.
- [6] D. Majewski, D. Liermann, P. Prohl, B. Ritter, M. Buchhold, T. Hanisch, G. Paul, and W. Wergen. The Operational Global Icosahedral-Hexagonal Gridpoint Model GME: Description and High-Resolution Tests. *Monthly Weather Review*, 130:319–338, 2002.
- [7] Ross J. Murray. Explicit generation of orthogonal grids for ocean models. *J. Comp. Phys.*, 126:251 – 273, 1996.
- [8] M. Rancic, R.J. Purser, and F. Mesinger. A global shallow-water model using an expanded spherical cube: Gnomonic versus conformal coordinates. *Quart. J. Roy. Meteor. Soc.*, 122:959 – 982, 1996.
- [9] Wiegers, K. E. *Software Requirements*. Microsoft Press, 1999.
- [10] Womack, B., Higgins, G. *Software Engineering Support of the Third Round of Scientific Grand Challenge Investigations. General Requirements Analysis*. NASA/CT Internal Document, 2002.

Part XII

Infrastructure Utilities: Performance Profiling

1 Authors, target codes and review team

Authors: David Neckels, Jim Rosinski

Review Date: TBD

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CAM-EUL, CLM	Rosinski
CAM-FV, PSAS	da Silva, Zaslavsky, Sawyer
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell, Young

Other Reviewers: DeLuca, Neckels, Jacob, Larson

2 Background

In order to efficiently optimize code, it is necessary to gain an understanding of how much processor time is spent in individual code sections. It is also sometimes necessary to obtain hardware information such as number of flops, cache hits/misses, and processor cycles, amongst other things. With this information in hand it is possible to identify code bottlenecks and re-arrange a program so that it calculates its results most efficiently.

The goal of this library is to provide a API for this type of profiling and to provide a reporting capability to nicely summarize the results.

2.1 Location

The profiler is a part of the Infrastructure. It is a low level class and does not have many dependencies. It will, however, be used by a number of the other classes in the library.

2.2 Scope

The profiler is responsible for code timings, and hardware profiling. It also will report the results of this survey.

3 Performance profiling requirements

PP1 Code section timing

The profiler will have a means to time individual code sections. An API will be provided so that a user may insert calls to the profiler at the beginning and end of a passage of interest.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP1.1 Named timers

Calls for recording time may be placed around numerous different code sections, and the library shall provide a means for distinguishing these separate calls. When the results of the timing are reported, each timer will be associated with its name.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP1.1.1 Named timer reset

A named timer may be reset at any time. All data will be cleared.

Priority:

Source: MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP1.2 Types of time

There are a number of different types of “time” that may be of interest to a profiler. These various types of time can be recorded. Each will be recorded separately and reported as such. The various types are enumerated below:

PP1.2.1 User time

Process user time can be recorded. The timer will provide a means to estimate and reconcile the extra time added by its usage.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes: It is desirable to offer a mechanism which removes the intrusiveness of the timing calls themselves; there are mechanisms to estimate the timing call overhead (WS).

PP1.2.2 System time

System time can be recorded.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP1.2.3 Wall clock time

Wall clock time can be recorded.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP2 Hardware counters

Most computers have special registers which keep track of information such as floating point operations, cycles, and cache hits/misses. The profiler will provide a high level interface for accessing these registers. Collection of the following data will be supported, where available:

- Floating Point Ops
- Cache Utilization
- Cache Misses
- Floating Point Unit Utilization
- Cycle Counts

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP3 Process granularity

The profiler will be able to operate within a nested structure of processes (e.g. from MPI process 3, thread 6).

Priority:

Source: CAM-FV, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP3.1 Process level

The profiler can be called safely from any process (MPI or other). It will identify and report which process it was called from.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP3.2 Thread level

The profiler can be called safely from any thread. It will identify and report which thread it was called from.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP4 Reporting

The profiler will provide a means to output the data collected. This output may be called at any point of execution, as many times as desired.

PP4.1 Log output

The profiler will write its output via the log.

Priority:

Source: MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP4.2 API retrieval

The profiler data will be accessible to the program by means of an API.

Priority:

Source: MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP4.3 Statistics

The profiler will be able to perform a variety of statistical analysis on its call data and report.

PP4.3.1 Thread statistics

A named timer can perform analysis across threads and report the minimum amount of time spent in any thread, the maximum, and the mean.

Priority:

Source: MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP5 Call deactivation

Calling profiling routines typically has a performance impact and should not be done during production runs. Therefore there will be some mechanisms to disable the profiling calls.

PP5.1 Compile deactivation

The library can be turned into stub calls at compile time.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

PP5.2 Runtime deactivation

A flag may be set so that calls to the profiling library do no more than check this flag and return.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes: It is particularly useful to be able to do some timing early on in a run to predict time to completion and then have the timing switch itself off.

Part XIII

Infrastructure Utilities: Log

1 Authors, target codes and review team

Authors: David Neckels, Shep Smithline, Jim Rosinski

Review Date: TBD

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji, Smithline
HIM	Hallberg
CAM-EUL, CLM	Boville
CAM-FV, PSAS	da Silva
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell

Other Reviewers: DeLuca, Neckels, Larson

2 Background

Output data from programs usually consists of a combination of numerical data, stored in formats such as NetCDF, visual data, and diagnostics. The log utility is intended to organize diagnostic output. Many programs use simple fortran `write` statements, and the output resulting from such statements in the multiprocessor environment is unpredictable and inconsistent at best. The utility tries to make this behavior predictable across computing environments. It also attempts to organize the diagnostic output so that searches and filters may be easily constructed.

2.1 Location

This utility is part of the infrastructure. This is an extremely low level object and will be used by most of the other objects in the project. It will thus not depend on the other objects in the library save perhaps the machine model.

2.2 Scope

The log will be for diagnostic output. The bandwidth is assumed to be moderate to small, i.e. it will not be used to output large streams of numerical model output data.

LG1 Log requirements

LG2 Interface characteristics

The log shall provide a simple interface with a minimal number of steps needed to begin use. The main messaging interface function shall be as close to the fortran `write` statement as possible.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

LG2.1 Fortran write interface

The log shall provide an interface that supports the fortran `format` notation. Therefore the log will be able to print out strings, reals, integers, arrays, and other built in fortran types.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS , MIT

Status: Proposed

Verification: Code Inspection

Notes:

LG2.2 Printf style interface

The log shall have a printf style interface so built in datatypes may be printed using the flexible format of `printf`.

Priority: 3

Source: MIT

Status: Proposed

Verification: Code Inspection

Notes:

LG3 Log states/levels

Each log message will have a level attached to it so that setting certain run states will allow turning certain classes of output on and off.

Priority:

Source: POP(desired), MIT

Status: Proposed

Verification: Code Inspection

Notes:

LG4 Output medium

The output from the log shall be to files.

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes:

LG5 Process organization

The library will allow output from different MPI processes to be distinguished and/or grouped together. Output from these processes can be written to some set of groupings of files, (e.g. one file per process, one file for all processes).

Priority:

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, POP, CICE, PSAS, MIT

Status: Proposed

Verification: Code Inspection

Notes: Output to one file should be sequentialized in a sensible manner (WS).

LG6 Flush command

The API will provide a flush command that will force the output to appear in the file(s) upon call.

Priority: 1

Source: CAM-EUL, CAM-FV, CLM, CCSM-CPL, PSAS , MIT

Status: Proposed

Verification: Code Inspection

Notes:

Part XIV

Infrastructure Utilities: I/O

1 Authors, target codes and review team

Authors: Leonid Zaslavsky, Arlindo da Silva, Michael Young, Mark Iredell

Review Date: 16 May, 2002

Target Codes	Reviewers
GFDL-SPEC, BGRID, MOM4	Balaji
HIM	Hallberg
CAM-EUL, CLM	Craig
CAM-FV, PSAS	da Silva
POP, CICE	Jones
WRF	Michalakes
MIT-REG, MIT-CPL, ADJ	Hill
NSIPP-ATM, NSIPP-OCN	Suarez
NCEP-ATM, SSI	Iredell, Young

Other Reviewers: DeLuca, Neckels, Jacob

Earth system modeling applications require efficient and robust tools for input and output of structured and unstructured gridded data, as well as observational data streams. Interfaces and methods provided by ESMF should allow reading and writing of data in several standard formats as well as support efficient internal data representations (see ESMF General Requirements [12], Section 8.1.3). The ESMF IO is supposed to provide a unified interface for input and output of high level ESMF objects such as Fields. The system is expected to automatically detect file formats at runtime, and to output data in a variety of formats, with the possibility of creating companion metadata files. Other file IO functionalities, such as writing of error and log messages and input of configuration parameters from an ASCII file, are not covered in this document. These will be the subject of the ESMF Log Requirement Document and the ESMF Control Requirement Document [?, ?].

1.1 I/O architecture

We use the experience of the WRF [?] and FMS [?] projects in defining the ESMF I/O architecture that is efficient, flexible, end-to-end, and package neutral. Our principles will be:

- Define a standard unified I/O interface and API covering ESMF-supported data models.
- Provide efficient implementation of this API for multiple data formats supported by the ESMF.

1.2 Data models

Earth system models use a variety of discrete grids to maintain information about fields in continuous space, as well as observations. The primary ESMF codes employ finite-difference and finite-volume grids, spectral grids, unstructured land-surface grids, and ungridded observational networks.

Fields within a model component are frequently defined on the same physical grid and are decomposed in memory in an identical fashion; that is, they share a distributed grid. They form a *bundle of fields* defined on the same distributed grid. The gridded data are supported by three ESMF elements: *PhysGrid* element for physical grids, *DistGrid* element for distributed grids, and *Fields* class for fields ([?], [?], [?]).

ESMF I/O will support input/output of data defined on all ESMF supported grids and location streams ([?], [?]). For the purpose of this document, we will consider data belonging to three broad categories:

Structured Gridded Data. A *structured grid* is one on which the relationship between gridpoints can be derived from their indices, without the need for an explicit map. A simple example is fields defined on a rectangular lat/lon grid.

Unstructured Gridded Data. For the more general *unstructured grid* the relationship between gridpoints cannot be derived from their indices, and the specification of an explicit map is necessary. An example is a *catchment grid* used by some land-surface models.

Observational Data on location streams. As defined in the *Physical Grid Requirements*, a location stream contains a list of locations

describe the measurements. Each observation is associated with a spatial point or region. A neighbor relationship is not defined for observations.

As we have already mentioned, logically rectangular grids are naturally represented by multi-dimensional arrays. The two latter data models can be represented as one-dimensional arrays of structures with each structure containing information about location, field values associated with this location, and a list of neighbors, if relevant.

1.3 Metadata. ESMF metadata conventions

Metadata is data about a digital object, “structured data about the data”. The metadata is usually provided by the creator or distributor of the object, and often either accompanies the object or is embedded in the file header. As such, metadata can be very useful as the basis for information storage and retrieval systems, as well as for utilization of the data within Earth Science models. The information about the object provided by metadata allows optimization of resource allocation and organization of storage and retrieval of data. In parallel computing, such a knowledge may be especially important.

If metadata are provided, the files may be either *self-described* or *co-described*, depending on the fashion in which metadata are allocated.

A self-described file contains in its header all metadata necessary to provide a unique interpretation of the file content assuming certain conventions.

A co-described file is accompanied by a metadata file. The metadata file provides a unique interpretation of the data file content under certain conventions.

It is assumed the metadata can be rapidly read by a corresponding API without reading an entire content of the data file. Some data files may contain complete description of their content, but the way data are represented might not allow rapid extraction of metadata. To make such a file co-described, its metadata could be extracted and allocated to a companion metadata file.

Some file formats that we discuss below, such as NetCDF and HDF, are organized according to well-defined rules. Their structures and APIs enable (but do not require) creation of self-described files. By narrowing the definitions, conventions enable a complete and unique description of each dataset.

We assume that the NetCDF conventions for climate and forecast metadata, “CF conventions”, will serve as a basis for ESMF metadata conventions. NetCDF Climate-Forecast Metadata Conventions [?] narrow definitions of NetCDF, an array-oriented data format and a library for gridded data [?], to allow a unique and complete description of gridded data used in geoscience. CF conventions specify standard dimensions, such as date or time (\hat{t}), height or depth (z), latitude (y), and longitude (x), and specify standard units for these dimensions and other quantities.

We expect ESMF metadata conventions to be based on the CF-conventions, and to cover fields defined on both structured and unstructured grids, as well as observational data. Unlike the CF conventions which are tightly associated with NetCDF, the ESMF conventions are supposed to be format neutral, and cover all of the ESMF data formats. These extensions will become the ESMF standard, and will be enforced by the ESMF I/O subsystem. However, the specification of ESMF metadata is optional, and users desiring not to specify any metadata should be able to do so.

1.4 Data formats

Several standard formats are currently used in Earth Science modeling for input/output of data:

NetCDF Network Common Data Form (NetCDF) is an interface for array-oriented data access. The NetCDF library provides an implementation of the interface. It also defines a machine-independent format for representing scientific data. Together, the interface, library, and format support the creation, access, and sharing of scientific data. The NetCDF software was developed at the Unidata Program Center in Boulder, Colorado. See [?]. In geoscience, NetCDF can be naturally used for representation of fields defined on logically rectangular grids. NetCDF use in geosciences is specified by CF conventions mentioned above [?].

To the extent that data on unstructured grids (or even observations) can be represented as one-dimensional arrays, NetCDF can also be used to store these data. However, it does not provide a high-level abstraction for this type of data.

DODS The Distributed Oceanographic Data System is a system that allows access to data over the internet. DODS is created and supported by Unidata Program Center in Boulder, Colorado. See [?]. DODS enables an implementation of NetCDF-client libraries that permits remote access to data through the Internet.

HDF The Hierarchical Data Format (HDF) project provides interface, software and file formats for scientific data management. The HDF software includes I/O libraries and tools for analyzing, visualizing, and converting scientific data.

HDF is developed and supported at the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign. There are two different HDF formats, HDF (4.x and previous releases) and HDF5. These formats are completely different and *not* compatible. See [?], [?].

HDF Scientific Data Sets API allows efficient operating with multi-dimensional arrays. Although HDF SDS itself does not provide a way to represent high-level abstractions for data on unstructured grids and observational data sets, HDF-based applications, such as HDF-EOS do so in HDF-EOS Point Point Structure.

HDF-EOS The Hierarchical Data Format - Earth Observing System (HDF-EOS) is the scientific data format standard selected by NASA as the baseline standard for the Earth Observing System (EOS). HDF-EOS is an extension of HDF and uses HDF library calls as its underlying basis. Version 4.1r1 of HDF is used. The library and tools are written in C language and a Fortran interface is provided. See [?].

HDF-EOS can be used for different data models within ESMF. Regular gridded data are supported by HDF-EOS Grid Structures, while HDF-EOS Point Structure covers unstructured grid and observational data.

GRIB GRIdded Binary (GRIB) is the standard gridded data format from the World Meteorological Organization (WMO). GRIB is a general purpose, bit-oriented data exchange format. Most NWP centers use GRIB for all the files produced from its analyses and forecasts. Since the GRIB standard does not specify standard API, NWP centers use a variety of software to process GRIB files.

The GRIB format used in ESMF shall be configurable, and shall allow the creation of files which conform to the NCEP standard usage.

IEEE Binary Streams A natural way for a machine to represent data is to use a native binary data representation. There are two choices of ordering of bytes (so-called *Big Endian* and *Little Endian*), and a lot of ambiguity in representing floating point data. The latter, however, is specified, if IEEE Floating Point Standard 754 is satisfied ([?], [?]). It is desirable to be able to use efficient native representation, and optionally provide ESMF metadata on a companion file using for example XML [?].

GrADS The Grid Analysis and Display System (GrADS) is popular visualization software widely used by the earth science modeling community (<http://grads.iges.org/grads/>). GrADS can read COARDS compliant NetCDF and HDF files, as well as IEEE binary and GRIB files provided an appropriate companion metadata file is provided (in GrADS parlance these are referred to as *control files*). Files produced by the ESMF are intended to be GrADS readable, and the ESMF shall produce GrADS control files upon request.

BUFR Binary Universal Form of Representation of the meteorological data (BUFR) is a self-descriptive format for observational data transmission introduced by the World Meteorological Organization [?]. The form and content of data contained in a BUFR message are described within BUFR message itself. In addition, BUFR provides condensation, or packing of data.

The BUFR is a table-driven code since the Data Description Section contains a sequence of data descriptors referring to a set of predefined and internationally agreed tables. Thus, instead of writing all detailed definitions within a message, one will just write a number identifying a parameter with its descriptions. The BUFR format used in ESMF shall be configurable, and shall allow the creation of files which conform to the NCEP standard usage, in which the predefined tables are contained in the file.

Modern data management approaches could potentially provide significant advantages in manipulating data and have to be carefully studied. For example, ESMWF has created and employed relational-database based Observational Data Base (ODB) software [?]. However, such complex data management systems are beyond the scope of the basic ESMF I/O.

1.5 Parallel I/O

The future development of ESMF IO facility will require further optimization with an expected increase in IO amount over the next few years. Two major factors contributing to the increase in I/O intensity are:

- Enhancement in model resolution;
- Increase in I/O frequency.

We also expect significant increase in amount of satellite data, although amount of I/O related to observational data is not comparable with amount of gridded I/O which drives our performance analysis.

There are two aspects of parallel IO:

- How dataset distributed among multiple processors can be written to a single file efficiently;
- How single file can be distributed across multiple physical discs and IO channels.

There are several possibilities to perform IO in parallel ([?]):

Single-threaded IO. A single process acquires all the data and writes them out. The features of hardware and OS are used to distribute the data over multiple channels and, possibly, to multiple disks.

Multithreaded, multi-fileset IO Many processes write to multiple independent files. These files may be assembled later. Since each of the processes operates with its file logically independently, we can again rely on the hardware and OS to operate concurrently with multiple channels and multiple disks.

Multithreaded/Single-fileset IO. Many processes write to a single file. Although this approach is the most desirable one, its implementation is the most complicated. Since it requires concurrent access to the file by multiple processes, it can be implemented within the ESMF I/O only when such functionality is provided by underlying I/O library.

Multithreaded IO offers a simple way to stripe the data across as many IO channels and disk channels as are available [?, ?, ?]. Parallel IO implemented in GFDL ([?]) supports parallel writing to single or multiple files. It supports NetCDF and binary data formats.

1.6 Synchronous and asynchronous IO

ESMF shall provide an asynchronous option for all ESMF IO models. It shall provide async read and write operations, the capability to wait on individual or groups of I/O operations, and query functions for the state of an operation.

1.7 Location

Input/Output (IO) is part of the ESMF Infrastructure. It will provide efficient utilities to input/output gridded data and observational data to and from the disk. A standard API will allow manipulation of multiple standard formats.

1.8 Scope

ESMF IO is meant to be used for standard API and underlying implementation, providing input and output of gridded data and observational data streams to and from the disk in multiple standard formats. I/O with different levels of parallelism have to be provided.

2 IO requirements

The following are requirements for IO operations and are in addition to the applicable general ESMF requirements (see ESMF General Requirements document, section 8.1.3).

IO1 General IO requirements

IO1.1 Establish ESMF metadata conventions

A set of metadata conventions on gridded data on structured and unstructured grids, as well as observational data has to be determined within ESMF. The ESMF metadata conventions shall be based on the NetCDF Climate-Forecast (CF) Metadata Conventions [?], extending it for unstructured gridded data as well as for observational data defined on location streams.

Priority: 1

Source: DAO, NCEP-GSM, NCEP-SSI, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT

Status: Proposed

Verification: Document inspection

IO1.2 Provide automatic generation of metadata

As an option, provide automatic generation of metadata for gridded data on structured and unstructured grids, as well as observational data covered by ESMF metadata conventions. This requirement shall cover all supported formats.

Priority: 1

Source: DAO, NCEP-GSM, NCEP-SSI, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT

Status: Proposed

Verification: Unit test

IO1.3 Provide generation of companion metadata file

Upon request, a companion metadata file, including the specified ESMF metadata, shall be provided for all supported formats.

Priority: 1

Source: DAO, NCEP-GSM, NCEP-SSI

Status: Proposed

Verification: Unit test

IO1.4 Provide generation of GrADS control file

Upon request, a companion GrADS file, including the specified ESMF metadata, shall be provided for IEEE binary and GRIB files.

Priority: 1

Source: DAO, NCEP-GSM, NCEP-SSI

Status: Proposed

Verification: Unit test

IO2 IO requirements for supporting different data formats

IO2.1 Reading and writing netCDF files for structured gridded data

ESMF has to provide interface and software to read and write netCDF files for structured gridded data covered by ESMF metadata conventions.

Priority: 1

Source: DAO, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT

Status: Proposed

Verification: Unit test.

IO2.2 Reading and writing netCDF files on unstructured grids

ESMF has to provide interface and software to read and write netCDF files for data on unstructured grids covered by ESMF metadata conventions.

Priority: 2

Source: DAO, NSIPP, MIT

Status: Proposed

Verification: Unit test.

IO2.3 Reading and writing netCDF files for observational data

ESMF has to provide interface and software to read and write netCDF files for observational data covered by ESMF metadata conventions.

Priority: 2

Source: DAO, NSIPP, MIT

Status: Proposed

Verification: Unit test.

IO2.4 Reading and writing netCDF files using DODS

ESMF has to provide interface and software to read and write netCDF files for data covered by ESMF metadata conventions over the Internet, using DODS.

Priority: 1

Source: DAO, NSIPP, MIT

Status: Proposed

Verification: Unit test.

Notes: The I/O ought to be extensible to other URI based systems.

IO2.5 Reading and writing HDF4 files for structured gridded data

ESMF has to provide interfaces and software to read and write HDF4 files for structured gridded data covered by ESMF metadata conventions.

Priority: 1
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.6 Reading and writing HDF4 files for data on unstructured grids

ESMF has to provide interfaces and software to read and write HDF4 files for data on unstructured grids covered by ESMF metadata conventions.

Priority: 1
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.7 Reading and writing HDF4 files for observational data

ESMF has to provide interfaces and software to read and write HDF4 files for observational data covered by ESMF metadata conventions.

Priority: 1
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.8 Reading and writing HDF5 files for structured gridded data

ESMF has to provide interfaces and software to read and write HDF5 files for structured gridded data covered by ESMF metadata conventions.

Priority: 2
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.9 Reading and writing HDF5 files for data on unstructured grids

ESMF has to provide interfaces and software to read and write HDF5 for data on unstructured grids covered by ESMF metadata conventions.

Priority: 2
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.10 Reading and writing HDF5 files for observational data

ESMF has to provide interfaces and software to read and write HDF5 files for data for observational data covered by ESMF metadata conventions.

Priority: 2
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.11 Reading and writing HDF-EOS files for structured gridded data

ESMF has to provide interfaces and software to read and write HDF-EOS files for structured gridded data covered by ESMF metadata conventions.

Priority: 1
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.12 Reading and writing HDF-EOS files for data on unstructured grids

ESMF has to provide interfaces and software to read and write HDF-EOS files for data on unstructured grids covered by ESMF metadata conventions.

Priority: 1
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.13 Reading and writing HDF-EOS files for observational data

ESMF has to provide interfaces and software to read and write HDF-EOS files for observational data covered by ESMF metadata conventions.

Priority: 1
Source: DAO
Status: Proposed
Verification: Unit test.

IO2.14 Reading and writing GRIB files for structured gridded data

ESMF has to provide interfaces and software to read and write GRIB files for structured gridded data covered by ESMF metadata conventions.

Priority: 1
Source: NCEP-GSM, NCEP-SSI
Status: Proposed
Verification: Unit test.

Notes: Although GRIB1 is currently in operational use at NCEP, it is proposed that ESMF will support GRIB2 format assuming transition in the next two years.

IO2.15 Reading and writing GRIB files for data on unstructured grids

ESMF has to provide interfaces and software to read and write GRIB files for data on unstructured grids covered by ESMF metadata conventions.

Priority: 3

Source:

Status: Proposed

Verification: Unit test.

Notes: Although GRIB1 is currently in operational use at NCEP, it is proposed that ESMF will support GRIB2 format assuming transition in the next two years.

IO2.16 Reading and writing native binary files

ESMF has to provide interfaces and software to read and write native binary files for structured gridded data, data on unstructured grids, and observational data covered by ESMF metadata conventions, for each machine accompanied by XML metadata for data covered by ESMF metadata conventions, assuming that the computer architecture supports the IEEE 754 Floating Point Standard.

Priority: 2

Source: DAO, NCEP-GSM, NCEP-SSI, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT

Status: Proposed

Verification: Unit test.

Notes: XML metadata should be specified. Possibility of BIG ENDIAN and LITTLE ENDIAN byte ordering should be taken into account. A byte swap and specified byte style facility is needed (MIT).

IO2.17 Reading and writing BUFR files

ESMF has to provide interfaces and software to read and write BUFR files for observational data covered by ESMF metadata conventions.

Priority: 1

Source: DAO, NCEP-SSI (milestone)

Status: Proposed

Verification: Unit test.

Notes: Limitations on types of data, etc.

IO3 Parallel I/O requirements

IO3.1 Single-threaded IO of distributed data

ESMF should provide single-threaded IO of distributed data when a single process acquires all the data from other processors and writes them to the disk or read data from the disk and passes them to other processes.

Priority: 1

Source: DAO, NCEP-GSM, NCEP-SSI, NSIPP, CAM-EUL, CLM, CCSM-CPL, MIT

Status: Proposed

Verification: Unit test.

IO3.2 Multi-threaded IO of distributed data to a multiple files

ESMF should provide multi-threaded IO of distributed data when each process reads data from and writes data to an independent file.

This include fast reading and writing independent temporary files by processors using high-speed disks.

Priority: 1

Source: DAO, NCEP-SSI, NSIPP, MIT

Status: Proposed

Verification: Unit test.

IO3.3 Multi-threaded read of distributed data from a single file

ESMF should provide multi-threaded read of distributed data when each process reads data from the the same file.

Priority: 1

Source: NCEP-GSM, NCEP-SSI, DAO, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT

Status: Proposed

Verification: Unit test. **Note** The ESMF I/O may provide this functionality only when it is enabled by underlying library.

IO3.4 Multi-threaded write of distributed data to a single file

ESMF should provide multi-threaded write of distributed data to the same file.

Priority: 2

Source: NCEP-GSM, NCEP-SSI, DAO, NSIPP, CAM-EUL, CLM, CCSM-CPL, POP, CICE, MIT

Status: Proposed

Verification: Unit test. **Note** The ESMF I/O may provide this functionality only when it is enabled by underlying library - otherwise it should provide a status bit to indicate the operation is not possible i.e. it shouldn't crash!

IO3.5 Synchronous and asynchronous IO

ESMF shall provide an asynchronous option for all ESMF IO models. It shall provide async read and write operations, the capability to wait on individual or groups of I/O operations, and query functions for the state of an operation.

Priority: 2

Source: DAO, NSIPP

Status: Proposed

Verification: Unit test.

Reference Material

- [1] Earth system modeling framework. <http://www.esmf.ucar.edu>, 2002.
- [2] A. Arakawa. Computational design for long-term numerical integration of the equations of atmospheric motion. *J. Comp. Phys.*, 1:119–143, 1966.
- [3] V. Balaji. Parallel numerical kernels for climate models. In ECMWF, editor, *ECMWF Teracomputing Workshop*. World Scientific Press, 2001.

- [4] R. Heikes and D. A. Randall. Numerical Integration of the Shallow-water Equations of a Twisted Icosahedral Grid. Part I: Basic Design and Results of Tests. *Monthly Weather Review*, 123:1862–1880, 1995.
- [5] P.W. Jones. First- and second-order conservative remapping schemes for grids in spherical coordinates. *Monthly Weath. Rev.*, 127:2204–2210, 1999.
- [6] D. Majewski, D. Liermann, P. Prohl, B. Ritter, M. Buchhold, T. Hanisch, G. Paul, and W. Wergen. The Operational Gblal Icosahedral-Hexagonal Gridpoint Model GME: Description and High-Resolution Tests. *Monthly Weather Review*, 130:319–338, 2002.
- [7] Ross J. Murray. Explicit generation of orthogonal grids for ocean models. *J. Comp. Phys.*, 126:251 – 273, 1996.
- [8] M. Rancic, R.J. Purser, and F. Mesinger. A global shallow-water model using an expanded spherical cube: Gnomonic versus conformal coordinates. *Quart. J. Roy. Meteor. Soc.*, 122:959 – 982, 1996.
- [9] Wiegers, K. E. *Software Requirements*. Microsoft Press, 1999.
- [10] Womack, B., Higgins, G. *Software Engineering Support of the Third Round of Scientific Grand Challenge Investigations. General Requirements Analysis*. NASA/CT Internal Document, 2002.

Part XV

Glossary

IO1 Glossary

This glossary defines terms used in Earth system modeling to describe parallel computer architectures, grids and grid decompositions, and numerical and computational methods. While some of the concepts in the glossary may eventually appear as computational objects, many will not. The goal here is not to define a framework design or an object model but simply to achieve a common language.

Accumulator A facility for collecting and averaging data values. Generally accumulators are associated with temporal averaging, although they might be associated with other weighted averaging operations.

Address space A standard term to refer to the memory seen by a computer program that it can write to directly using simple language primitives.

Alarm An event that occurs at a particular time (or set of times). It is like an alarm on a real alarm clock except that in order to determine whether it is "ringing", an alarm is "read" by an explicit application action.

Addressable node A set of processors that are capable of addressing the same set of blocks of physical memory.

Application A coherent computational entity run as a single executable or set of communicating executables. It typically consists of a set of interacting components.

Background grid A background grid associates each point in a location stream with a location on a grid. A single grid cell may contain zero or more location stream points.

Bundle A bundle refers to a set of fields that are associated with the same physical grid and distributed in a similar fashion across the same physical axes. Fields within a bundle may be staggered differently and may have different dimensions.

Calendar interval A period of time specified in calendar-based units that may be used to increment or decrement time instants. One year and three months is an example of a calendar interval. Since mathematical operations involving calendar intervals may be ambiguously defined – for example, incrementing January 31 in the Gregorian calendar by one month – default behavior must be carefully specified.

Cell A physical location that is specified by both its extent (vertices) and nominal central location, and is associated with a single integer index value or a set of integer index values (e.g. (i) for 1-d, (i,j) for 2-d, (i,j,k) for 3d).

Clock A clock tracks the passage of time and reports the current time instant, like a real clock. However, most clocks used in ESMF components have a key difference to a real clock. Clocks in an ESMF component are generally stepped forward by the component, as an explicitly coded time step within the overall component.

Component A large-scale computational entity associated with a particular physical process or computational function, such as a land model. Components may be generic or user-supplied. See also gridded component, coupler component.

Compute resource Something that appears as a physical or virtual computer resource. Example of compute resources are a CPU, a network connection, a communication API, a protocol, a particular network fabric or a piece of computer memory.

Coupler component A component that includes all data and actions needed to enable communication between two or more other components.

Data dependency The property of a computational operator that defines the data indices required to perform the computation at a point. For instance, a forward differencing operation in X at (i, j) has a dependency on $(i + 1, j)$.

Data transpose Rearrangement of data arrays between two distributed grids sharing the same global domain.

Day of year The day number in the calendar year. January 1 is day 1 of the year. Day of year expressed in a floating point format is used to express the day number plus the time of day. For example, assuming a Gregorian calendar:

<u>date</u>	<u>day of year</u>
10 January 2000, 6Z	10.25
31 December 2000, 18Z	366.75

Distributed grid A distributed grid defines the decomposition of the global index space across the layout and methods on the indexed data.

Distribution The function that expresses the relationship between the indices in a distributed grid and the elements in a layout.

Domain decomposition The act of grid distribution: creating a layout; and associating gridpoints with the layout. The dimensionality of the domain decomposition is the dimensionality of the associated layout.

Exact The word exact is used to denote entities, such as time instants and time intervals, for which truncation-free arithmetic is required.

Exchange grid A grid whose vertices are formed by the intersection of the vertices of two overlying grids. Each cell in the exchange grid overlies exactly one cell in each grid of the exchange.

Exchange packets The data exchanged by components. Exchange packets may or may not contain contiguous data, and may contain both field and other forms of data.

Exclusive domain The set of indices whose data is exclusively and definitively updated by a particular PE.

Executable A parallel program that is under independent control by the operating system.

Export state The data and metadata that a component can make available for exchange with other components. This may be data at a physical boundary (e.g land-atmosphere interface) or in other cases, it might be the entire model state. See also restart state, import state.

Field A field is a physical quantity defined within a region of space. A field includes a grid and any metadata necessary for a full description of the field data.

Functionality Class A functionality class is a body that accomplishes a given function, such as I/O. It may contain several different classes or extend over multiple framework layers. Functionality classes are described in the ESMF Architecture Document.

Generic component A generic component is one supplied by the framework. The user is not expected to customize or otherwise modify it. See also user component.

Generic transform A generic transform is a operation supplied by the framework, for example, a method that converts gridded data from one supported physical grid and/or decomposition to another using a specified technique. See also user transform.

Global physical grid A global physical grid contains physical information about the entire, undecomposed domain. No distributed grid need be associated with a global physical grid.

Global domain The global range of indices of data points.

Global reduction Reduction operations (sum, max, min, etc.) on data defined on a distributed grid. See also global broadcast.

Global broadcast Scatter operations on data defined on a distributed grid. See also global reduction.

Grid The discrete division of space associated with a particular coordinate system. A grid contains all physical grid and memory organization information (via distributed grid and layout) required to manipulate fields, as well as to create and execute grid transforms.

Grid metrics Terms relating measurements in index space to physical grid quantities like distances and areas.

Grid staggering A descriptor of relative locations of scalar and vector data on a structured grid. On different staggered grids, vector data may lie at cell faces or vertices, while scalar data may lie in the interior. The staggered locations are often written in a notation like $(i + \frac{1}{2}, j + \frac{1}{2})$ to describe the offset of a corner with respect to the cell (i, j) .

Grid topology Description of data connectivities in index space.

Grid union The formation of a new grid by taking the union of the vertices of two input grids.

Gridded component A component that is associated with one or more grids. No requirements may be placed on the physical content of a gridded component's data or on the nature of its computations.

Halo The points in the data domain outside the local domain.

Halo update Halo points are associated with other PEs' local domains, and the halo update operation involves synchronization of some or all halo points with other PEs.

Import state The data and metadata that a component requires from other components in order to run. See also export state, restart state.

Index An integer value associated with a set of coordinates that describe a cell or location in physical space.

Index space The space implied by a set of indices. An index space has a defined dimensionality and connectivity.

Index space location A location within index space. A index space location may be fractional. See also physical location.

Layout A layout specifies a PE list, decomposition strategy (thread and process), and the dimensionality and connectivity of the decomposition. Multiple distributed grids may be defined per layout.

Local domain This includes the exclusive domain, as well as the points with whom the exclusive points have data dependencies.

Local physical grid The portion of a physical grid associated with a local domain.

Location stream A list of locations with no assumed relationship between these locations. The elements of a location stream are assumed to share the same data items and metadata, though some elements may have blank entries for particular data or attributes.

Logically rectangular grid A grid in which sequential indices are physically adjacent, and in which the extent of each index is independent of the other indices.

Loose bundle A loose bundle consists of fields whose data is not contiguous in memory.

Mask A field marking a span within a larger grid.

Memory domain The portion of memory associated with an local domain. The memory domain is always at least as large as the local domain.

Memory node A set of processors sharing equal flat access to a block of physical memory.

MPMD Multiple Program Multiple Datastream. Multiple executables, any of which could itself be an SPMD executable, executing independently within an application.

No-leap calendar Every year uses the same months and days per month as in a non-leap year of a Gregorian calendar.

Packed bundle A packed bundle is arranged so that field data is contiguous in memory.

Partition In a multi-threaded application, the subset of a computational domain that is associated with a logically independent sequence of operations. The logical independence requirement is so that partitions may be scheduled as separable concurrent tasks.

PE Short for processing element.

PE list A list of processor IDs associated with a component. See also layout.

Physical grid A physical grid contains a variety of information on the location in physical space and physical metrics (area, grid lengths, etc.) of various grid points.

Physical location The point in physical space to which data pertain.

Platform The processor hardware, operating system, compiler and parallel library that together form a unique compilation target.

Processing node A set of processors to which an operating system scheduler is capable of assigning to a single job.

Restart state The component data that is needed for an exact restart. This can include, in addition to a physical state, time information, static field data, metadata and control information.

Scheduler An operating system component that assigns system resources (processors, memory, CPU time, I/O channels, etc.) to executables.

Span The physical extent associated with a grid.

SPMD Single Program Multiple Datastream. A single executable, possibly with many components (representing for example the atmosphere, the ocean, land surface) executing serially or concurrently.

System time Time spent doing system tasks such as I/O or in system calls. May also include time spent running other processes on a multiprocessor system.

Time instant Generic name for an absolute time and date specification. A time instant is made up of a time and date and an associated calendar. It may include a time zone. "Jan 3rd 1999, 03:30:24.56s, UTC" is one example of a time instant.

Time interval A time interval is the period between any two time instants, measured in units, such as days, seconds, and fractions of a second, that are not associated with a specific calendar. Time intervals may be negative. The periods 2 days and 10 seconds, 86400 and 1/3 seconds and 31104000.75 seconds are all examples of time intervals. Mathematical operations such as addition, multiplication and subdivision can be applied to time intervals.

User component A component that is customized or written by the user. See also generic component.

User time Processor time actually spent executing a process's code.

User transform A user-supplied method that is used to extend framework capabilities beyond generic transforms.

Wall clock time Elapsed real-world time (i.e. difference between start time minus stop time).