1. Introduction	1
1.1. Release notes	
1.2. Brief history of ICM	
1.3. ICM distribution and support	
1.4. What can you do with ICM? (a program overview)	13
1.4.1. Graphics.	
1.4.2. Simulations.	16
1.4.3. Sequence analysis.	19
1.4.4. Modules of ICM.	
1.5. Notational conventions	
1.6. Common abbreviations	25
1.7. Getting started.	
1.7.1. ICM-shell.	
1.7.2. The first steps	
<u>1.7.2. The first steps</u>	
2. Reference Guide	31
2.1. ICM command line options.	
2.2. Command line editing.	
2.3. Graphics controls.	
2.4. Editing pairwise sequence–structure alignments.	
2.5. Constants.	
2.6. Subsets and index expressions.	
2.7. Molecule intro.	30
2.8. Selections	30
<u>Selection Types</u>	/10
Selection levels.	
<u>Examples</u>	
Select by number, range, name or pattern	
2.8.1. Object selection.	
2.8.2. Molecule selection.	
2.8.3. Residue selection.	
2.8.4. Atom selection.	
2.8.5. Free and all variables (v and V).	
2.8.6. Functions returning selections.	47 50
2.8.7. Finding contiguous residue ranges with the String function	
2.9. Arithmetics.	
2.9.1. Assignment	
2.9.2. Arithmetic operations.	
2.9.3. Logical operations.	
2.9.4. Comparison operators.	
2.10. Fly sector and some comments.	
2.10. Flow control	
<u>2.10.1. Loops</u>	
2.10.2. Conditional branching	
<u>2.10.3. Jumps</u>	59
2.11. ICM molecular objects	
2.12. Energy and Penalty Terms.	61

2. Reference Guide	
2.13. Integer shell parameters.	65
2.13.1. autoSavePeriod	65
2.13.2. defSymGroup	
<u>2.13.3. i out</u>	
<u>2.13.4, iProc</u>	66
2.13.5. maxColorPotential	
2.13.6. maxMemory	67
2.13.7. minTetherWindow.	
2.13.8. mnSolutions	
2.13.9. mncalls.	
<u>2.13.10. mncallsMC</u>	
<u>2.13.11. mnconf</u>	
2.13.12. mnhighEnergy.	
<u>2.13.13. mnreject</u>	
<u>2.13.14. mnvisits</u>	
<u>2.13.15. nLocalDeformVar</u>	
<u>2.13.16. nSsearchStep</u>	
<u>2.13.17. nProc</u>	
<u>2.13.18. randomSeed</u>	
2.13.19. segMinLength	
2.13.20. sequenceBlock	
2.13.21. sequenceLine	
<u>2.13.22. surfaceAccuracy</u>	
2.14. Real shell variables	
2.14.1. addBfactor	
2.14.2. alignMinCoverage.	
2.14.3. alignOldStatWeight	
2.14.4. axisLength	
2.14.5. cnWeight	
2.14.6. dcWeight	
2.14.7. densityCutoff.	
2.14.8. dielConst	73
2.14.9. dielConstExtern.	
2.14.10. drop	
<u>2.14.11. fogŠtart</u>	
2.14.12. gapExtension	
2.14.13. gapOpen	74
<u>2.14.14. hbCutoff</u>	
2.14.15. lineWidth	
2.14.16. mapSigmaLevel	75
<u>2.14.17. mcBell</u>	
2.14.18. mcJump	
2.14.19. mcShake	
2.14.20. mcStep	
2.14.21. mfWeight	

	<u>2.14.22. mimelDepth</u>	76
	2.14.23. mimelMolDensity.	76
	2.14.24. r out	
	<u>2.14.25. r_2out</u>	77
	2.14.26. resLabelShift	
	2.14.27. rsWeight	
	2.14.28. selectMinGrad	77
	2.14.29. selectSphereRadius.	
	2.14.30. shininess	
	2.14.31. ssThreshold	
	2.14.32. ssWeight	
	2.14.33. ssearchStep	
	2.14.34. surfaceTension	
<del>,</del>	2.14.36. temperature	
	2.14.30. tellfperature	
<del>,</del>	2.14.37. tololad. 2.14.38. tzWeight.	
	2.14.39. vicinity.	
	2.14.40. vwCutoff.	
	2.14.41. vwExpand	
	2.14.42. vwSoftMaxEnergy	
	2.14.43. waterRadius	
	2.14.44. wireBondSeparation.	
	2.14.45. xrWeight.	
	rical variables.	
	2.15.1.1 antiAlias	
	2.15.2.1 autoLink	
	2.15.3.1 bpmc	
	2.15.4.1 breakRibbon	
	2.15.5.1 bufferedOutput	82
	2.15.6.1 bug	
	2.15.7.1 caseSensitivity	82
	2.15.8.1 commands	
	2.15.9.1 confirm	83
	2.15.10.1 easyRotate	83
	2.15.11.1 info	
	2.15.12.1 minRedraw	
	2.15.13.1 neutralAcids.	
	2.15.14.1 out	
	2.15.15.1 print	
	2.15.16.1 readMolArom.	
	2.15.17.1 showAccessibility.	
	2.15.18.1 showMC	
	2.15.19.1 showMinSteps	
	2.15.20.1 showSpecialChar	
4	2.15.21.1 showSites	85

2. Reference	Guide
--------------	-------

2.15.22.1 showSstructure	85
2.15.23.1 showWater	
2.15.24.1 showTerms	
2.15.25.1 warn	
2.15.26.1 wrapLine	
<u>2.15.27.1 writeStartObjMC</u>	
2.15.28.1 xrUseHydrogen	86
2.16. String variables.	
2.16.1. s blastdbDir.	
2.16.2. s editor	
2.16.3. s entryDelimiter.	
2.16.4. s errorFormat	
2.16.5. s fieldDelimiter.	
2.16.6. s helpEngine.	
2.16.7. s icmhome	
<u>2.16.8. s inxDir</u> .	
<u>2.16.9. s icmPrompt</u>	
<u>2.16.10. s imageViewer</u>	
<u>2.16.11. s labelHeader</u> .	89
2.16.12. s lib	
<u>2.16.13. s logDir</u>	
<u>2.16.14. s_out</u>	00
2.16.15. s pdbDir	
<u>2.16.16. s projectDir</u> .	
2.16.17. s printCommand	
<u>2.16.18. s prositeDat</u>	
<u>2.16.19. s psViewer</u>	
<u>2.16.20. s reslib</u>	
2.16.21. s skipMessages : ignore specific error messages	01
2.16.22. s tempDir.	92
2.16.23. s translateString.	
<u>2.16.24. s_userDir</u> .	
<u>2.16.25. s usrlib (obsolete)</u>	92
2.16.26. s webEntrezLink.	
2.16.27. s webViewer.	92
2.16.28. s xpdbDir	93
<u>2.17. Preferences</u> .	
2.17.1. atomLabelStyle.	93
2.17.2. alignMethod	94
2.17.3. atomSingleStyle	
<u>2.17.4. dcMethod</u>	
2.17.5. electroMethod.	
2.17.6. errorAction	96 96
2.17.7. ffMethod.	
<u>2.17.7. inventou</u> <u>2.17.8. gcMethod</u>	
2.17.9. highEnergyAction.	
2.17.7. Inginency Action	

2. Reference Guide	
2.17.10. interruptAction	98
2.17.11. mfMethod	98
2.17.12. minimizeMethod	
<u>2.17.13. pdbDirStyle</u>	
2.17.14. rejectAction	
<u>2.17.15. resLabelStyle</u>	
2.17.16. ribbonColorStyle	
<u>2.17.17. ribbonStyle</u>	
<u>2.17.18. shineStyle</u>	
2.17.19. surfaceMethod	
<u>2.17.20. tzMethod</u>	
2.17.21. varLabelStyle.	
2.17.22. visitsAction	
<u>2.17.23. vwMethod</u>	
2.17.24. webEntrezOption.	
<u>2.17.25. wireStyle</u>	
<u>2.17.26. xrMethod</u>	
2.18. Tables (structures)	
<u>2.18.1. FILTER</u>	
<u>2.18.2. FTP.</u>	
<u>2.18.3. GRAPHICS</u>	
<u>2.18.4. GRID</u>	
<u>2.18.5. GROB</u>	
<u>2.18.6. GUI</u>	
<u>2.18.7. IMAGE</u> 2.18.8. LIBRARY	
<u>2.16.8. LIBKAK I.</u> 2.18.9. OBJECT	
<u>2.18.9. OBJECT</u> 2.18.10. PLOT	
<u>2.18.10. FLOT</u>	
2.18.12. WEBLINK	
<u>2.18.13. WEBAUTOLINK</u>	
2.19. Other shell variables	
<u>2.19.1. defCell</u>	
2.19.1. accFunction.	
2.19.3. gapFunction	
2.19.4. I out	
2.19.5. M out	
2.19.6. R out	
2.19.7. S out	
2.19.8. swissFields.	
2.19.9. readMolNames	
2.19.10. Named Atom/Residue/Molecule/Object Selections	121
<u>2.19.11. as out</u>	
2.19.12. as2 out	
2.19.13. Named Selections of Internal Variables (Dihedrals, Angles and Bonds)	
2.19.14. vs_out	122

2. Reference Guide	100
2.20. Commands	
<u>2.20.1. alias</u>	
2.20.2. align	
<u>2.20.3. assign</u>	
2.20.4. break	
2.20.5. build	
2.20.6. call icm script	
<u>2.20.7. center</u>	
<u>2.20.8. clear</u>	
2.20.9. color family of commands	137
2.20.10. compare: setting conformation comparison parameters for	the montecarlo
command	
2.20.11. compress	145
2.20.12. connect	146
2.20.13. continue	147
2.20.14. convert	147
2.20.15. copy	
2.20.16. crypt	
2.20.17. delete ICM shell objects.	
2.20.18. display	
2.20.19. elseif.	
2.20.20. endfor	
2.20.21. endif	
2.20.22. endmacro	
2.20.23. edit	
2.20.24. endwhile	
<u>2.20.25, exit</u>	
<u>2.20.26. find</u>	
<u>2.20.27. fix</u>	
2.20.27. <u>11X</u> 2.20.28. for	
<u>2.20.28.101</u> 2.20.29. fork	
<u>2.20.29. 101N</u>	
<u>2.20.30. iprinu</u> . <u>2.20.31. goto</u>	
<u>2.20.32. group</u> 2.20.33. gui	
<u>2.20.34. help</u>	
<u>2.20.35. history</u>	
<u>2.20.36. if</u>	
<u>2.20.37. keep</u>	
2.20.38. link internal variables of molecular object	
2.20.39. link residues to sequences and alignments.	
<u>2.20.40. list</u>	
2.20.41. list available sequence databases	
<u>2.20.42. load</u>	
2.20.43. ICM-shell macros	
2.20.44. make	

2. Reference Guide	
<u>2.20.45. minimize</u>	200
<u>2.20.46. menu</u>	
<u>2.20.47. modify</u>	
<u>2.20.48. montecarlo</u>	
<u>2.20.49. move</u>	
<u>2.20.50. pause</u>	
<u>2.20.51. plot</u>	
2.20.52. plot area: show matrix values with color.	
<u>2.20.53. print</u>	
<u>2.20.54. printf</u>	
<u>2.20.55. print image</u>	
<u>2.20.56. quit</u>	
<u>2.20.57. randomize</u>	
<u>2.20.58. read</u>	
<u>2.20.59. rename</u>	
2.20.62. rotate	
<u>2.20.65. set rainity of commands</u> 2.20.64. show	
<u>2.20.66. split</u>	
2.20.67. sprintf.	
2.20.68. store	
2.20.71. superimpose.	
2.20.75. undisplay	
2.20.76. unfix	
2.20.78. wait	
2.20.79. web	
2.20.80. web table: shows an icm table with a web browser	
2.20.81. while	
2.21. Functions	
<u>2.21.1. Abs</u>	
<u>2.21.2. Acc</u>	
<u>2.21.3. Acos</u>	
<u>2.21.4. Acosh</u>	
<u>2.21.5. Align</u>	
<u>2.21.6. Angle</u>	
<u>2.21.7. Area</u>	

2.21.8. Area contact matrix	
2.21.9. Asin	
2.21.10. Asinh	
2.21.11. Ask	
2.21.12. Atan	
2.21.13. Atan2	
2.21.14. Atanh	
2.21.15. Atom.	
2.21.16. Augment.	
2.21.17. Axis	
2.21.18. Bfactor.	
2.21.19. Boltzmann	
2.21.20. Box	
2.21.21. Bracket	
2.21.22. Cad.	
2.21.23. Ceil	
<u>2.21.24. Cell</u>	
2.21.25. Charge.	
2.21.26. Cluster.	
2.21.27. Color.	
2.21.28. Consensus	
2.21.29. Corr.	
<u>2.21.30. Cos</u>	
<u>2.21.31. Cosh</u> .	
<u>2.21.32. Count</u>	
2.21.33. Deletion	
<u>2.21.35. Detenon</u> <u>2.21.34. Det</u>	
2.21.35. Disgeo	
2.21.36. Distance	
2.21.37. Eigen	
2.21.38. Energy.	
2.21.39. Error.	
2.21.40. Exist.	
2.21.41. Existeny	
2.21.42. Extension	
2.21.43. Exp	
2.21.44. Field.	
2.21.45. User field from a selection.	320
2.21.46. File	321
2.21.47. Find.	
2.21.48. Floor	
<u>2.21.49. Getenv.</u>	
2.21.50. Gradient	
<u>2.21.50. Gradient</u>	
2.21.52. Histogram	
2.21.53. Iarray	

2.21.54. Iarray( stack ): numbers of visits for all stack conformations	
2.21.55. IcmSequence	
2.21.56. Index.	
2.21.57. Indexx	
2.21.58. Insertion.	
2.21.59. Integer	
2.21.60. Integral.	
2.21.61. Interrupt	
2.21.62. Label.	
2.21.63. Length	
2.21.64. LinearFit	
<u>2.21.65. Link</u>	
2.21.66. Log	
<u>2.21.67. Map</u>	
<u>2.21.68. Mass</u>	
<u>2.21.69. Matrix</u>	
<u>2.21.70. Max</u>	
2.21.71. MaxHKL	
<u>2.21.72. Mean</u>	
<u>2.21.73. Min</u>	
2.21.75. Mill. 2.21.74. Money	
<u>2.21.74. Moley</u> 2.21.75. Mod	
<u>2.21.75. Mou</u> 2.21.76. Mol	
<u>2.21.77. Name</u>	
<u>2.21.77. Name</u> . <u>2.21.78. Namex</u> .	
<u>2.21.78. Namex</u>	
2.21.79. Next. 2.21.80. Covalent neighbors of an atom	
2.21.80. Covarent neighbors of an atom.	
<u>2.21.82. Norm</u> 2.21.83. Obj	
<u>2.21.84. Occupancy</u>	
<u>2.21.85. Path</u>	
2.21.86. Pattern.	
<u>2.21.87. Pi</u>	
<u>2.21.88. Potential</u>	
<u>2.21.89. Power</u>	
<u>2.21.90. Probability</u>	
<u>2.21.91. Profile</u>	
<u>2.21.92. Putenv</u>	
<u>2.21.93. Radius</u>	
<u>2.21.94. Random</u>	
<u>2.21.95. Rarray</u>	
2.21.96. Real function	
2.21.97. Remainder function.	
<u>2.21.98. Replace</u>	
<u>2.21.99. Res</u>	353

2.21.100. Res(ali): from sequence positions in subalignment to residue selection	
2.21.101. Resolution	
<u>2.21.102. Rfactor</u>	
<u>2.21.103. Rfree</u>	
<u>2.21.104. Rmsd</u>	
<u>2.21.105. Rot</u>	
<u>2.21.106. Sarray</u>	
<u>2.21.107. Score</u>	
<u>2.21.108. Select</u>	
2.21.109. Sequence	359
2.21.110. reverse complement dna sequence function	360
2.21.111. Sign	360
2.21.112. Sin	360
2.21.113. Sinh	361
2.21.114. Site.	
2.21.115. Smiles	
2.21.116. Smooth	
2.21.117. Sql	
2.21.118. Sqrt	
2.21.119. Sphere	
2.21.120. Split	365
2.21.121. Srmsd.	
2.21.122. String	
2.21.123. Chemical formula	
2.21.124. Sstructure.	
<u>2.21,124, 050 detute</u> <u>2.21,125, Sum</u>	
<u>2.21.125. Sum</u> . 2.21.126. Symgroup.	
2.21.120. <u>Syngroup</u> 2.21.127. Table	
2.21.127. Table. 2.21.128. Converting alignment into a table	
2.21.129. Extracting parameters of stack conformations	512
2.21.129. Extracting parameters of stack comormations.	
<u>2.21.130. Tali</u> 2.21.131. Tanh	
2.21.132. Tensor	
<u>2.21.195.1011poruture</u>	
<u>2.21.134. Time</u> .	
<u>2.21.135. Tolower</u>	
<u>2.21.136. Torsion</u>	
<u>2.21.137. Toupper</u>	
<u>2.21.138. Tr123</u>	
<u>2.21.139. Tr321</u>	
<u>2.21.140. Trace</u>	
<u>2.21.141. Trans</u>	
<u>2.21.142. Transpose</u>	
<u>2.21.143. Trim</u>	
<u>2.21.144. Turn</u>	
2.21.145. Type	379

2. Reference Guide	
2.21.146. Unix	380
2.21.147. Value	381
2.21.148. Vector	381
2.21.149. Version	381
2.21.150. Volume	382
2.21.151. View	383
2.21.152. Warning : the ICM warning message	
2.21.153. Xyz : atom coordinates and surface points	384
2.22. Macros	
2.22.1. buildpep: Building peptides from a sequence	385
2.22.2. calcBindingEnergy: estimates electrostatic, hydrophobic and entropic	
binding terms	386
2.22.3. calcDihedral4atoms: calculate a torsion angle defined by four atoms	
2.22.4. calcDihedralAngle: calculate an angle between two planes in a molecule	386
2.22.5. calcEnsembleAver: Boltzmann average the energies of the stack	
<u>conformations</u>	
2.22.6. calcMaps: calculate five energy maps and write them to files	
2.22.7. calcPepHelicity: calculate average helicity of a peptide from movie frames	387
2.22.8. calcProtUnfoldingEnergy: rough estimate of solvation energy change upon	
unfolding	
2.22.9. calcRmsd: calculate three types of Rmsd between protein conformations	
2.22.10. calcSeqContent.	
2.22.11. icmCavityFinder: analyze and display cavities.	
2.22.12. dsCellBox: displays crystallographic unit cell	
2.22.13. dsCell: cell and crystallographic neighbors.	
2.22.14. dsCharge: one of many ways to show charge residues	
2.22.15. dsChem : chemical style display. 2.22.16. dsConsensus: 3D display of conserved residues	
2.22.17. dsCustom: extended display and property-coloring	
2.22.17. dsCustom: extended display and property-coloring	
2.22.19. dsCustonirun macro for molecular display.	
2.22.19. dsDistance: display distances between two selections.	
for binding.	
2.22.21. dsEnergyStrain: analyzing energy strain in proteins	
2.22.22. dsEnergyStrain1.	
2.22.23. icmPmfProfile	
2.22.24. dsPrositePdb	
2.22.25. ds roster du	
2.22.26. dsSeqPdbOutput : visualize the sequence similarity search results	
2.22.27. dsSkinLabel.	
2.22.28. dsSkinPocket and dsSkinPocketIcm	
2.22.29. dsStackConf.	
2.22.30. dsVarLabels	
2.22.31. ds3D.	
2.22.32. dsWorm	
2.22.33. dsXyz : display.	

<u>2.22.34. findFuncMin</u>	
2.22.35. findFuncZero.	
2.22.36. nice	
<u>2.22.37. cool</u>	
2.22.38. homodel	399
2.22.39. makeIndexChemDb	399
2.22.40. makeIndexSwiss	
2.22.41. makePdbFromStereo: restore 3D coordinates from a stereo picture	
2.22.42. mkUnigPdbSequences.	
<u>2.22.43. plot2DSeq</u>	
2.22.44. plotSegDotMatrix.	400
$\frac{2.22.44}{2.22.45}$ plotSeqDotMatrix2.	
<u>2.22.46. plotBestEnergy</u>	
2.22.47. plotOldEnergy	
2.22.48. plotFlexibility	
2.22.49. plotCluster	
<u>2.22.49. plotCluster</u> 2.22.50. plotMatrix	
<u>2.22.51. plotRama</u>	
2.22.52. plotRose 2.22.53. plotSeqProperty	
<u>2.22.55. protseqProperty</u>	
2.22.55. prepSwiss.	
2.22.56. printFast	
<u>2.22.57. printMatrix</u>	
2.22.58. printPostScript.	
2.22.59. printTorsions	
2.22.60. refineModel: globally optimize side-chains and anneal the backbone	
<u>2.22.61. regul</u>	
<u>2.22.62. rdBlastOutput</u>	
<u>2.22.63. rdSeqTab</u>	
2.22.64. readPdbList	
<u>2.22.65. remarkObj</u>	
2.22.66. searchPatternDb	
2.22.67. searchPatternPdb	
2.22.68. searchObjSegment	405
2.22.69. searchSeqDb	
2.22.70. searchSeqPdb	405
2.22.71. searchSeqPdb	
2.22.72. searchSeqSwiss	
2.22.73. setResLabel	406
<u>2.22.74. sortSeq</u>	406
<u>2.22.75. undsCharge</u>	406
2.22.76. makeSimpleModel.	406
2.22.77. makeSimpleDockObj	406
2.22.78. searchSeqProsite	406
2.23. Files.	

	407
2.23.1. macro. A collection of ICM macros.	
2.23.2. startup. ICM startup file.	
2.23.3. startCheck script.	
2.23.4. foldbank.db.	408
2.23.5. Bank of protein folds (foldbank.seg).	
2.23.6. Atom codes (icm.cod)	
2.23.7. Bond angle bending and improper torsion deformation parameters (icm.bbt).	
2.23.8. Bond stretching parameters (icm.bst)	
2.23.9. Conformational stack (*.cnf).	410
2.23.10. Distance restraint types ( icm.cnt or *.cnt )	410
2.23.11. Distance restraints (*.cn)	
2.23.12. Graphics objects (*.gro)	
2.23.13. ICM HTML help file ( icm.htm )	
2.23.14. Hydrogen bonding types ( icm.hbt )	411
2.23.15. Hydration parameters ( icm.hdt ).	411
2.23.16. Configuration file (icm.cfg).	412
2.23.17. Colors ( icm.clr ).	412
2.23.18. Electron density map (*.map)	
2.23.19. MC simulation movie (*.mov).	
<u>2.23.20. ICM-object (*.ob</u> )	414
2.23.21. Residue library ( icm.res or *.res )	
<u>2.23.22. Object Variables (*.var)</u>	
2.23.23. Multidimensional variable restraint types (icm.rst or *.rst)	415
2.23.24. Multidimensional variable restraints (*.rs)	
<u>2.23.25. A sample *.col file</u>	417
<u>2.23.26. A sample *.tab file</u>	
2.23.27. Torsion parameters (icm.tot)	
2.23.28. Van der Waals parameters ( icm.vwt )	418
2.23.29. Protein databank file ( or *.ent )	418
2.23.30. Sequence ( *.seq *.pir *.gcg *.msf * )	419
2.23.31. ICM-sequence file (*.se).	
<u>2.23.32. ICM–alignment file</u>	
2.23.33. ICM all-file: a file with multiple icm objects.	
2.23.34. Residue comparison table ( icm.cmp or *.cmp )	421
2.23.35. Protein profiles (*.prf)	422
2.23.36. Integer array (*.iar).	
2.23.37. String array (*.sar)	
<u>2.23.38. Matrix (*.mat)</u>	422
2.23.39. Numerical data (real arrays) (*.rar).	423
3. User's guide	425
<u>3.1. ICM–shell</u>	
3.1.1. How to get help	425
3.1.2. Customization.	
3.1.3. How to write a nice demo with menus to impress the boss	426
3.1.4. How to boost learning process while reading the ICM manual	426

3. User's guide	
3.1.5. How to get the list of the command words	427
3.2. ICM graphics	427
3.2.1. How to learn the ICM molecular graphics in 30 seconds	427
3.2.2. How to make a nice high-resolution image	
3.2.3. How to rotate one molecule around its own center of mass	
3.2.4. How to annotate a molecular image in the graphics window	
3.2.5. How to save and print the generated image.	429
3.2.6. How to change the color of the graphics window background	
3.2.7. How to return a molecule to the center of the graphics window	
3.2.8. How to color atoms according to their B-factors	
3.2.9. How to color residues according to their hydrophobicities	
3.2.10. How to color residues according to their accessibilities	430
3.2.11. How to color atoms according to their charges	
3.3. Structure analysis.	
3.3.1. How to optimally superimpose two 3D structures	
3.3.2. How to optimally superimpose without the residue alignment	
3.3.3. How to make a Ramachandran plot	
3.3.4. How to display hydrogen bonds	
3.3.5. How to identify atoms or residues at the molecular interface	
3.3.6. How to identify torsions at the molecular interface	
3.3.7. How to calculate packing density	435
3.3.8. How to perform a principal component analysis	
3.3.9. How to calculate a dihedral angle	
3.3.10. How to print a table of the torsion angles	
3.3.11. How to build a hydrophobicity profile.	
3.3.12. How to display and characterize protein cavities	
3.4. Sequence, searches and alignments.	
3.4.1. How to search all Prosite patterns in your sequence	
3.4.2. How to find a fragment in the PDB database (obsolete)	
3.4.3. How to identify binding pockets.	
3.4.4. How to find a similar fold or topological motif in the PDB database	
3.4.5. How to generate a non-redundant list of PDB sequences	
3.4.6. How to merge several pdb files.	
3.4.7. How to compile a database of protein secondary structures and their folds	
3.4.8. How to search headers of the PDB entries.	
3.5. Energetics and electrostatics.	
3.5.1. How to plot the distance dependence of a van der Waals interaction	443
3.5.2. How to calculate the electrostatic free energy by the REBEL-method	443
3.5.3. How to evaluate the pK shift.	
<u>3.5.4. How to evaluate the binding energy</u>	
3.5.5. How to calculate an ensemble average	444 1 1 5
3.5.6. How to evaluate nelicity of a peptide from the BPMC simulation	
<u>3.6. Manipulations with molecules</u>	
<u>3.6.1. How to build new object from a sequence</u> 3.6.2. How to quickly convert a pdb file into an ICM-object	440
<u>5.0.2. How to quickly convert a pdb file into an ICM-object</u>	447

### 3. User's guide

3.6.3. How to prepare a PDB structure for energy calculations (regularization)	447
3.6.4. How to create a new molecule or a residue for the ICM residue library	448
3.6.5. How to modify an ICM-object	448
3.6.6. How to merge two ICM-objects	451
3.6.7. How to make a hybrid model from several pdb files	452
3.6.8. How to generate a series of intermediates between the two given structures	<u>.</u> 453
3.6.9. How to reconstruct a structure from a published stereo picture	
3.7. Animation	454
3.7.1. How to rotate and zoom in a script	454
3.7.2. How to make a molecular movie from a Monte Carlo trajectory	455
3.8. Transformations and symmetry.	457
3.8.1. Main concepts and functions	457
3.8.2. How to generate symmetry related molecules	
3.8.3. How to find and display rotation/screw transformation axis	458
3.8.4. How to combine several transformations	
3.8.5. How to build a helix from the two contacting monomers	
3.9. Maps and factors.	460
<u>3.9.1. How to manipulate with structure factors</u>	460
3.9.2. How to calculate phases of reflections given a 3D model and a cell	460
3.9.3. How to automatically place a fragment into density	
<u>3.10. How to plot</u>	
3.10.1. How to make a simple plot $y=F(x)$	
3.10.2. How to plot a histogram	461
3.10.3. How to make a 3D-surface plot of a 2D-function	462
3.10.4. How to create a new graphics object of a specific shape	
3.10.5. Flexible peptide docking	462
3.11. How-to: Docking and Virtual Ligand Screening	
3.11.1. Docking and virtual ligand screening. Overview.	
3.11.2. How-to: Ligand docking simulations.	
3.11.3. How-to: Virtual Ligand Screening	
<u>3.12. Example scripts</u>	
3.12.1. How to predict 3D structure of a peptide from its sequence	
3.12.2. How to perform local flexible docking of two protein molecules	475
3.12.3. How to perform an explicit flexible docking of two simplified protein	
molecules.	
3.12.4. How to build a model by homology	476
4. References	177
4.1. General literature references.	411 177
4.1. General interactive references. 4.2. The main description of the ICM method.	
4.2. The main description of the ICM method. 4.3. ICM algorithms.	479 1 <b>7</b> 0
4.4. ICM applications	
4.5. Credits.	
<u></u>	

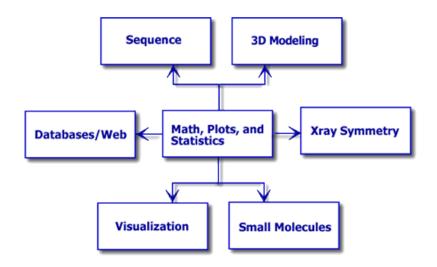
5. Glossary		
	<b>Y</b>	
	alignment	
	<u>all</u>	
	alpha helix	484
	<u>amber</u> .	
	append.	
	<u>atom</u>	
	axis	
5.2. F	3	
01212	base	
	ball	
	beta	
	boundary element.	
	BPMC	
5.3. 0		
	- <u>cavity</u>	
	charge	
	<u>coil</u>	
	column	
	current map	
	current object	
	current table	
	<u>command</u>	
	<u>comp_matrix</u>	
	<u>conf</u>	
	<u>cpk</u>	
<u>5.4. I</u>	<u>)</u> <sup>–</sup>	
	database	
	Depth-cueing, or fog	
	distance	
	distance geometry.	
	disulfide bond	
	drestraint.	
<i></i>	drestraint type	
<u> 3.3. E</u>	<u>3-H</u>	
	ecepp.	
56 9	<u>fasta</u>	
<u></u>		
	<u>site</u>	
	<u>grob</u> hbond– hydrogen bonds	
	svariablei, or ICM-shell variable	
	integer.	
	label	
	logical	
	<u>10 510 44</u>	······································

### 5. Glossary

	macro	495
	map	495
	matrix	
	MIMEL	
	mmff .	
	mol	
	mol2.	
	more	
	<u>movie</u>	
	mute	
	<u>only</u>	
	pattern.	
	png	
	pdb or Protein Data Bank	
	peptide bond	
	profile	
	prosite	
	<u>REBEL</u>	
	<u>real</u>	
	regularization.	
	residue	
	<u>rgb</u>	
	<u>ribbon</u>	
	<u>script</u>	
	sequence	
	segment	
	(ICM)-shell	
	<u>skin</u>	506
	smiles	507
	<u>sln</u>	507
	<u>stack</u>	507
	stick	508
	string	
	structure factor (factor).	
	surface area.	
5.7. T		
	table	
	Pairwise table expressions:	
	table subsets:	
	tether	
	transformation vector.	
5.8. U-2		
5.0. 0-1	unique	
	virtual atoms and variables.	
	volume.	
	vrestraint.	
	<u>vicsualin</u>	

5. Glossary		
	vrestraint type	
	wire	
	xstick	
	ZEGA	
Index		

# 1. Introduction



ICM stands for Internal Coordinate Mechanics and was first designed and built to predict low energy conformations of biomolecules. ICM also is a programming environment for various tasks in computational structural biology, sequence analysis and rational drug design. The original goal was to develop algorithms for energy optimization of several biopolymers with respect to an arbitrary subset of *internal coordinates* such as bond lengths, bond angles torsion angles and phase angles. The efficient and general global optimization method which evolved from the original ICM method is still the central piece of the program. It is this basic algorithm which is used for peptide prediction, homology modeling and loop simulations, flexible macromolecular docking and energy refinement. However the complexity of problems related to structure prediction and analysis, as well as the desire for perfection, compactness and consistency, led to the program's expansion into neighboring areas such as graphics, chemistry, sequence analysis and database searches, mathematics, statistics and plotting.

The original meaning became too narrow, but the name was kept. The current integrated ICM shell contains hundreds of variables, functions, commands, database and web tools, novel algorithms for structure prediction and analysis into a powerful, yet compact program which is still called ICM. The seven principal areas are centered around a general core of shell–language and data analysis and visualization.

### 1.1. Release notes

Last Updated: Aug 19,2002 .

In this section we keep track of all the latest changes in different modules of ICM.

Selecting atoms and residues by user-defined field values Aug 18,2002

User can define one atom field (see set field and Field (*selection* [*i\_field*])) and three residue fields. Now one can also select atoms or residues using the Select function with arguments looking like this: "u>3." for atoms. For residues "u", "v" and "w" values can be used for three user fields. Example:

 $Select(a_/* "u>3.")$ # select residues with 1st user field greater than 3.

#### improvement of split grob. Limit size of generated grobs 11,2002

Now you can specify the minimal size of a grob generated as a result of the the split command, e.g. split gg 500.0

#### extracting grob vertices with certain color Aug 11,2002

Grob(grob { from\_R, to\_R, from\_G, to\_G, from\_B, to\_B }) function which returns a grob with selection of points of the source grob. The selected points will have colors between the RGB values provided in the 6-dim. array of limits (from 0. to 1.).

#### set grob label s\_Label Aug 10,2002

Assign a string label to a grob.

Name( *object\_type* select ) August 10, 2002 Returns string array of names of the selected objects.

Nof( *object\_type* select ) August 10, 2002 Counts the selected objects. Convenient for GUI scripting (see the icm.gui file).

sort objects and sort molecules July 30, 2002 You can now resort objects and molecules by a user field or by molecular mass. See sort objects or sort molecules.

Access to the warning messages from shell: the Warning function July 24, 2002 Example:

if Warning() message = Warning(string)

#### Permissions for tables: set property args table1 table2 .. July 24, 2002

tables can now have a number of properties: read only, edit the cells only, etc. See the set property command.

#### Error(string) Text of each error message and its numerical error code July 24, 2002

Now all icm error messages you can extract the full text of the message. Example:

if Error errorMessage = Error(string)
print i\_out # the numerical code of the message

#### Remove unwanted characters. Trim( s\_sourse, s\_listOfOkCharacters ) July 18, 2002

#### Revert order in arrays July 18, 2002

Iarray( {1 2 3} reverse) # returns {3 2 1}

Sarray( S reverse) Rarray( R reverse)

Unique atomic ordering for chemical compounds: the make unique command. July 15, 2002

Fast refinement into electron density (the gcMethod preference ). July 15, 2002

explicit sequence type for the Sequence(*s\_seq* [nucleotide | protein ]) function. April 22, 2002

option box in make grob map April 10, 2002 It controls addition of the surrounding box to the contoured map.

averaging two matching arrays: Mean (R1 R2) April 9, 2002

minimize loop command and making loop optimization movies April 9,2002 The build model command may not be able to find a perfectly matching loop. Two sorts of problems may appear: the imperfections of the loop attachments and the clashes of the loop to the body of the model. They are resolved with the minimize loop *i\_loopNumer* command.

window={minValue,maxValue} range for color mapping April 8, 2002 Now the display and color commands in which you color a selection by a matching array of values, can be told what range of values the array is clamped to. A single digital value can also be used as a color with the window option. Example:

color ribbon a\_/ Bfactor(a\_/ simple) window=-0.5//2. color ribbon a\_/14 0.8 window=-0.5//2. # single value coloring

This command will clamp Bfactor(a\_/ simple) values which are normally around zero, but may range from large negative values to large values, to the [-0.5,2.] range.

**relative B-factor. April 3, 2002** In read pdb now the average B-factor for all non-water atoms is calculated for X-ray objects. The normalized B-factor ( $(b-b_av)/b_av$ ) is now returned by the Bfactor(as\_simple) function. This is much better for coloring ribbons by B-factor since it only depends on ratios to the average. We recommend to use:

color ribbon a\_/ Trim(Bfactor( a\_/ simple ),-0.5,3.)//-0.5//3. # or color a\_// Trim(Bfactor( a\_// simple ),-0.5,3.)//-0.5//3. # for atoms

This scheme will give you a full sense of how bad a particular part of the structure is.

**save tab- and comma delimited tables.** April 3, 2002 s\_fieldDelimiter=", " or "\t" will be now affecting write table You can also specify it in the command line like this:

write table separator="," # or separator="\t", etc. write table header separator="," # will save column names

Atom chirality April 1, 2002 (not a joke!) Atom chirality flags have been introduced. Each atom can have a chirality flag set to:

• zero - non-chiral center

1. Introduction

- 1 left topoisomer
- 2 right topoisomer
- 3 racemic mixture of both isomers

Here is a short summary of new commands and functions associated with chirality:

i/o	chirality flag is read from a mol file or ICM object.		
set chiral as_flag	set chiral <i>as_flag</i> set chirality flag to a selection		
a_//X[1230LRB]	select chiral atoms, e.g. $a_//X1$ for left atoms, $a_//X$ is equivalent to $a_//X123$		
show as_	shows a new Xi field with the chirality value		
l_racemicMC	this flag allows to switch chirality in icm global optimization		
Dot-separated chemi	ical formula for multiple molecules March 30,2002		

Use String( dot, selection ) to get a dot separated chemical formula ( String( all, selection ) will give the total formula).

# Setting, storing and extracting user-defined properties of atoms, residues, molecules and objects March 22. 2002

User can now store real properties of atoms, residues, molecules and objects directly in the object.

- To set those properties, use the set field selection [number=i] property command.
- To extract the property use Field( *selection*, *i\_fieldNumber* )
- To clear use the set field clear *selection* command (or explicitly set it to zero).

Level	Max.Nof_fields		example
Atom	1	set	field a_//c* Mass(a_//c*)
Residue	3	set	field a_/trp 1. number=2
Molecule	16	set	<pre>field a_W Random(1.,10.,Nof(a_W)) number=12</pre>
Object	16	set	field a_*. Rarray(Count(Nof(a_*.)))
Version()	function to repo	ort the	e license codes. March 12. 2002 The Version function now returns

one-character license codes (e.g. " H D ").

```
Version()
Version(full)
```

**parsing more PDB object types. March 9, 2002** ICM now recognizes more object types as specified by the EXPDTA card of a pdb-file, see Type ( os\_2 ):

"ICM" ready for energy calculations. Those objects are either built in ICM or converted to the ICM-type. "X-Ray" determined by X-ray diffraction "NMR" determined by NMR "Model" theoretical model (watch out!) "Electron" determined by electron diffraction

"Fiber" determined by fiber diffraction

"Fluorescence" determined by fluorescence transfer

"Neutron" determined by neutron diffraction

"Ca-trace" upon reading a pdb, ICM determines if an object is just a Ca-trace.

"Simplified" special object type for protein folding games.

**Passing arguments to an ICM script: icm** -a string option March 8, 2002 You can initialize s\_icmargs string with the icm -a 'args' option. Example:

```
% icm -s -a 'al.sdf a2.sdf'
s_icmargs
a1.sdf a2.sdf
args = Split(s_icmargs)
for i=1,Nof(args)
   print args[i]
endfor
```

**String( { as\_| rs\_| ms\_| os\_} } i\_number ) March 1, 2002** Function String ( selection ) is further extended to print i-th element of a selection. It is convinient in scripts. For atoms it will also show the full information about an atom, rather than only the ranges of atom numbers.

Detecting if pmf parameters (Nof(pmf). Feb 16, 2002 Nof(pmf) function reports the number pmf types.

Converting selections to strings. String(  $as_| rs_| ms_| os_)$  function extended. Feb 14, 2002 Now you can make a selection at any level and the String function will return a selection string which can be stored in variables, arrays and tables. Note, that the chemical formula is now returned by the String( all  $as_)$  or String( dot  $as_)$  function (keyword may be the last arugment as well).

Accumulating numbers of visits in the compress stack command. Feb 12, 2002 Now when you compress the conformation stack, the number of visits after the compression will be the total number of visits from all.

"Ca-trace" objects. Feb 9, 2002 Frequently Ca-trace-only objects in PDB create problems in automated database analysis scripts. Now this type is identified upon readin and it can be checked with the Type( a\_2) command. Also if you try to convert the "Ca-trace" object to the ICM type, the convert command returns an error. If the object is mixed and contains both all atom models and Ca-traces, the Ca-traces will be skipped.

**improvements of the convert command. Feb 6, 2002** Previously when the convert command was applied to an object with *missing* atoms in the backbone, the convert command had problems and generated ICM objects deviated from their pdb-templates significantly. This problem is now fixed. Also, ICM reports the deviation from the template and it is returned in the r\_out command.

Unification of the  $a_//$  and  $a_1.//$  selections. Feb 12, 2002 There was a discrepancy between short forms of selections of all residues and all atoms. Now for the selections have the same level (e.g.  $a_//$  or  $a_1.//$  select all atoms in the current or the first object, respectively)

setting number of visits and energy for stack conformations. Jan 29, 2002 You can now specify the number of visits and the energy manually in the store conf command.

#### 1. Introduction

writing sequence groups into fasta/msf files. Jan 27, 2002 groups of sequences can be now saved or shown to fasta or msf files.

**uncompressing bzip2 on the fly. Jan 25, 2002** Seemless treatment of the .bz2 compressed files is added. E.g.

```
read sequence "aaa.seq.bz2"
364255 sequences loaded:
```

**setting b-factors at residue level. Jan 20, 2002** Previously b-factors were always assigned at the atom level. Now, if you have a real array of residue energies (e.g. returned by the Energy(rs\_) function, you can assign b-factors at the residue level with a simple command:

set bfactor rs\_R\_resBfactors

#### reading .sdf .mol files Jan 18, 2002

Features from multiple sdf files are now extracted into the S\_out string array upon reading. E.g.

```
read mol "aaa.mol"
logp = Rarray(Field(S_out, "logp",1,"\n")) # rarray of logp values
can_numbers = Trim(Field(S_out, "cas_rn",1,"\n"))
```

#### selecting water Jan 17, 2002

Selecton a\_W selects both water molecules and deuterated waters. This may now replace less rigorous a\_w\* expressions in macros. E.g. delete a\_W This selection is equivalant to the longer Mol( a\_/hoh,dod ) expression.

#### selecting aromatic atoms. Jan 15, 2002

Selection  $a_//R$  will select aromatic atoms. It selects all heavy atoms connected by aromatic bonds and hydrogens attached to them.

#### moving selections from one object to another Jan 11, 2002

Frequently a selection of atoms, residues of molecules need to be tranferred from one object to another. Now there is a function Select( selection, os\_targetObject ). Example :

```
read pdb "2ins"
copy a_ "bb"
aa = a_1.2,4,5//c*
bb = Select(aa,a_bb.) # moves the selection
```

This is useful in macros.

#### surface energy Jan 5, 2002

The surface energy can now be calculated using grid potentials.

#### preference GRID.gcghExteriorPenalty Jan 3, 2002

van der Waals grid energy *outside* van der Waals grids "gc" and "gh" used to be positive to push molecules back to the grid box. Now there is a preference: GRID.gcghExteriorPenalty which can be set to "repulsive" (the previous mode) or to "zero" to have no field outside the box.

#### zero-size rarrays and iarrays can be created. Jan 1, 2002

Rarray(0) and Iarray(0) now allow to create arrays of zero size.

#### convertObject macro extended to convert a selected set of molecules Dec 26, 2001

#### s\_skipMessages allows to suppress printing of selected error and warning messages Dec 15, 2001

s\_skipMessages is new shell string variable which contains a list of error/warning codes which you want to suppress in your icm output. Despite suppression of the message, the error is still generated and can be returned with the Error(string) function in i\_out. Example:

s\_skipMessages = "[234][5243]"

Macro to write sequences in the alignment order. Nov 28, 2001 Frequently you may want to renumber your object (which may contain omitted loops and ends) according to a full protein sequence. Now you can do it automatically with the

align number *ml\_seq\_* 

command.

#### Macro to reorder sequences in the alignment in the read order. Nov 26, 2001

The alignment changes the input order of sequences. To keep the order the same in an alignment, you may now use the

reorderAlignmentSeq( ali\_) macro from the \_bioinfo file.

See the Align function for more details.

#### Macro to write sequences in the alignment order. Nov 24, 2001

A tiny macro wrSeqAli (file \_macro) allows to write sequences in the alignment order.

**Type( seq\_2 ) : type of a sequence Nov 22, 2001** find out the sequence type with the Type( seq\_2 ) function.

#### normal distribution : Random( mean std N "gauss") Nov 14, 2001

Generate an array with normally distributed numbers with the Random() function, e.g. Random( 0., 1., 10, "gauss")

set grob reverse : inverting normals in grobs, Nov 9, 2001 Permanently change direction of normals of one or all grobs allows to change direction of lighting as well as influence the sign of Volume(grob). Now one can:

#### 1. Introduction

set grob reverse # all grobs
set g\_ reverse # a specific grob

full pairwise alignment with arbitrary positional weights, Nov 8, 2001 We introduced a new command to set custom sequence residue areas (e.g. set area lcrn\_m {0.3,1.,1.3,0.8,..}). This 'areas' can be used by the Align(seq1,seq2,area) function for weighted pairwise alignment. Example:

set area lcrn\_m 0.
set area lcrn\_m Random(0. 2. Length(lcrn\_m))

#### New macros, Nov 7, 2001

icmCavityFinder to find closed cavities icmPocketFinder to find open cavities potential of mean force , Nov 4, 2001

A new energy term "mf" is added. It allows to establish interactions according to pairs of atoms types. This term can be used both for chemical superposition and pharmacophore modeling (e.g. you know several ligands to the same receptor and want to derive the pharmacophore model), or for scoring the docking solutions in virtual ligand screening. Example:

```
read pmf "ident.pmf" buildpep "his ; ala trp" \# two molecules: his and ala-trp fix v_//omg montecarlo "mf"
```

#### read pdb sequence chain case, Nov 2, 2001

Originally PDB was case insensitive. However, in some entries it is used to specify different chains, e.g. A and a in 1fnt . ICM now copes with this misdemeanor of PDB.

#### ~/.icm/user\_startup.icm file, Oct 28, 2001

#### font control in GUI terminal: Oct 25, 2001

icm.cfg file can now be define the **font size** in the terminal window. The specification may look like this:

XTermFont \*-fixed-medium-\*-\*-\*-24-\*

#### gui simple: keep the original terminal window and start gui, Oct 22, 2001

Previously the gui command under Unix/Linux disabled the original terminal window and created a new one inside the GUI window. This new window had problems with font controls. Now you can keep the original window and start GUI.

```
% icmgl
gui simple
    # or
% icmgl -G
```

#### Distance(alignment1 alignment2 exact), Oct 21, 2001

a measure of differences between two alignments (see Distance(*ali,ali,exact*)) to evaluate the influence of alignment parameters on alignments, or to evaluate differences between a trial alignment and a 'golden' standard.

#### display origin, Oct 19, 2001

allows to display coordinate frame. Also: undisplay origin

#### set v\_ reverse , Oct 18, 2001

modify internal coordinates while keeping the remote part of the molecule unmoved

#### copy a\_ tether , Oct 18, 2001

the copy command is extended to allow simultaneous copying and imposing tethers to the copy. Also, now the default name of the copy-object is a\_copy.

#### refineModel macro, Oct 17, 2001

refineModel macro from the \_macro file allows to refine models resulting either from homology model (in this case it is supposed to be tethered to the target object) or any other source. It performs iterative energy refinement with appropriate annealing of the tethers. If you only want to improve the predicted side chains, use refineModel 0 yes

#### a\_//Z selection for tether-target atoms, Oct 18, 2001

Previously one could select tethered atoms ( $a_//T$ ) but not the atoms to which this atoms are tether. Now it can be done with the  $a_//Z$  selection.

#### montecarlo reverse, Oct 16, 2001

a new reverse option of the montecarlo command allows to make a more intellegent random move of a molecule fragment surrounded by other static molecules. It prevents the movement of the whole molecule if a backbone angle at N-terminus is changed.

#### Average gradient amplitudes of residues Oct 14, 2001

Gradient( rs\_ | as\_ ) now returns not only the array of gradient lengths for atoms ( e.g. Gradient( $a_{//*}$  ) ) but also returns averaged gradient lengths for each selected residue if the selection is of residue level (e.g. Gradient( $a_{/*}$ )). This can be used to color the ribbon or to direct a montecarlo procedure.

#### Easy coloring of 3D models by space averaged alignment strength. Oct 8,2001

see ribbonColorStyle

ribbonColorStyle = "reliability"

allows to color a ribbon model by alignment strength with 3D gaussian averaging of the conservation signal. selectSphereRadius controls the radius of the averaging.

#### Reading NMR entries with CRYST data.

Some new NMR entries in PDB have illegal CRYST ORIGX etc. data. Previously ICM complained about them. Now, it the experiment is NMR, the crystallographic symmetry fields are skipped.

#### Sampling with probabilities according to b-factors . Oct 5,2001

Now you can use a new bfactor option of the montecarlo command to sample 'hot' parts of structure with higher probabilities. The relative frequences are taken from the b-factors of the atoms belonging to the mc-variables. Example:

```
buildpep "ala his trp glu" # default b-factor=20
set bfactor a_/2 1000. # make 2nd his hot
montecarlo bfactor
```

To preserve the old bfactors, save them before the simulation and restore after. E.g.

b\_old = Bfactor(a\_//\*) # save .. set bfactor a\_/10:20 200. montecarlo bfactor .. set bfactor a\_//\* b\_old # restore

#### Detection of missing fragments in pdb-files. Oct 3,2001

ICM detects chain missing residues according to the differences between SEQRES sequence and the residues with coordinates and returns the total number of missing residues in the *i\_out* system variable. E.g.

```
read pdb "lamo.a/"
make sequence a_1.1 # sequence lamo_1_a extracted
if(i_out>1) then
  read pdb sequence "lamo" # sequence lamo_a read
  a=Align(lamo_a lamo_1_a)
  build model lamo_a a_1.1 a # patch the missing fragments
endif
```

**Renumber residues in fragments. Oct 3, 2001** previously only renumbering of the whole protein chain was allowed Example:

```
read pdb "lcrn"
align number a_/10:20 101 # renumber a fragment starting from 101
```

#### Total masses of residues, molecules, objects. Oct 3, 2001

The same extension as the one below was applied to the Mass function (e.g.  $Mass(a_/*)$  returns masses of residues.

#### Total charges of residues, molecules, objects. Sep 14, 2001

Charge ( os\_ | ms\_ | rs\_ | as\_ [formal | mmff] ) now returns the array of total charges for each selected unit (e.g. Charge( $a_{/*}$ ) returns an array of total charges of each selected residue)

#### Numbers of atom arrays in residues, molecules, objects. Sep 13, 2001

Nof(  $a_*$ .\* atom ) # returns an iarray of number of atoms in each molecule

#### Cheminformatics: Modifications of non-ICM objects Aug 10, 2001 Example:

```
read mol "scaffold.mol" # contains r1 atoms
read mol "bblocks.sdf" # contains blocks with attachment points atoms
modify a_1.//r1 a_2.//a # build a molecule
build hydrogen
set type mmff
set charge mmff
convert
```

**Modeling: tzMethod = "weighted"** to use b-factors of atoms for weighting tethers. Aug 7, 2001 method of imposing and calculating tethers. The three alternatives are the following

- 1. "simple" : equal weight tethers to 3D points (the old one)
- 2. "weighted" : individual weights are calculated from atomic B-factors by dividing 8\*PI<sup>2</sup> by the B-factor value. All the weights additionally are multiplied by the tzWeight shell parameter.
- 3. "z\_only" : tethers are imposed only in the Z-direction towards the target Z-coordinate. These type of tethers allow to pull a molecule into a z-plane. This may be useful if you are trying to generate a flat projection of a three-dimensional molecule.

Modeling: tzMethod = "z\_only" to tethers to the z-plane of other atoms. Aug 1, 2001

#### Distance restraints improvements. July 17, 2001.

Target distance allows to have a target not in the middle of upper and lower bounds.

#### v\_//T and v\_//F selections

#### New maps and binding site predictions- Sep. 01.

Two commands have been extended:

- make map potential (no terms). calculates new map m\_atoms. Can be used on pdb-objects
- make grob map exact r\_contourLevel . allows to specify absolute (exact) contouring level

A new map with gaussian envelope around atoms can be quickly built for ICM and non–ICM objects and used to find *semi–closed* pockets. If contoured at 0.3 it gives

buildpep "his arg"
make map potential Box( a\_ 3.) solid

#### 1. Introduction

```
make grob m_atoms 0.2 exact # contours near vw-raduis.
display cpk
display g_atoms
```

To identify pockets: try different contouring levels and split the grobs.

#### bug fixes:

modify phase branch convert a\_ "new name"

### 1.2. Brief history of ICM

ICM author's heads in italic



Ruben Max

The first lines of ICM were born in 1985 out of a desire to build a general method for predicting the structure of complex biological macromolecules and their complexes. I formulated a set of requirements for a program for molecular mechanics in a full set of internal coordinates, and started working on the internal coordinate algorithms and the Fortran code of the first program blocks. By 1991 the batch parameter files were replaced by a command language and an interactive shell that looked quite similar to the current version of ICM, and the molecules started to follow commands and sample the energy minima.

**Max Totrov** and I extended or rewrote most parts of ICM from 1991 to 1994. By 1993 several people (Alexey Mazur, Mikhail Petukhov, and Dmitry Kuznetsov) had also contributed to the fortran version of ICM, however their contributions did not survive in the current version of the program. Alexey pursued the development of molecular dynamics in internal coordinates which was first formulated and tested in a series of papers in 1989 and, later, branched out of ICM.

The all–C version of ICM emerged in 1994 as a result of a full rewrite. Some features were lost, but more were gained. **Serge Batalov** joined the development of the program in the fall of 1994, about the time Molsoft was founded. Max and I together with the new Molsoft developers keep ICM strong, clean and healthy.

### 1.3. ICM distribution and support

ICM is being developed, distributed and supported by Molsoft, LLC.

If you have any problem with our programs, please contact Molsoft via e-mail:

support@molsoft.com

Please indicate the platform, the version of the program, and do not forget all the necessary files to reproduce it. Some of the commands or functions described in this manual belong to specific modules and

are not available in the ICM-main program.

### 1.4. What can you do with ICM? (a program overview)

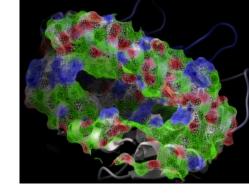
Let us go through the short overview of ICM applications.

### 1.4.1. Graphics

### Versatile surface and structure views to elucidate protein function

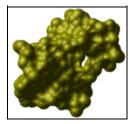
The views include

- binding and active site surfaces with mapped properties
- automatic identification and views of cavities and open binding pockets
- electrostatic surfaces



### Analytical molecular surface (skin)

The contour–buildup algorithm calculates the smooth and accurate analytical molecular surface in seconds. This surface can be saved as a geometrical object, saved as a vectorized postscript file.



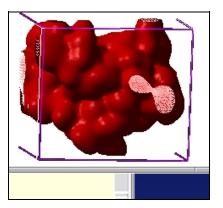
The skin is used in the REBEL algorithm to solve the Poisson equation, as well as in the molecular surface analysis routines (e.g. a projection of physical properties on the receptor surface ).

Also ICM can build and draw a solvent-accessible surface (see surface) and

\* a gaussian molecular density which can be contoured at different

levels and to generate different smooth molecular envelopes and enclosed pockets and cavities:

```
make map potential Box( a_ 3.)
make grob m_atoms exact 0.5 solid
display g_atoms smooth
```

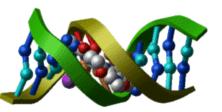


### Schematic representations of DNA and RNA

PDB entry: 101d

ICM command:

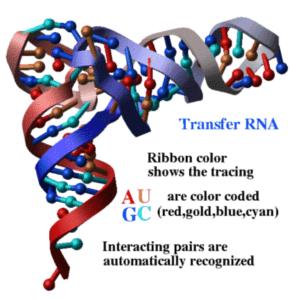
nice "101d"



PDB entry: 4tna

#### ICM commands:

```
nice "4tna"
color ribbon a_N/* Count(Nof(a_N/*))
```



### **Complex combined representations**

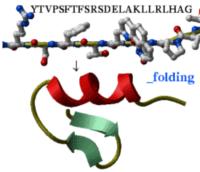
THIS MESS DOES NOT REFLECT THE LACK OR ARTISTIC TASTE
3236

Simplified molecular representations are built automatically (e.g. the protein-dna complex is shown with one command: nice "ldnk"). You can combine different types of molecular representations with solid or wire geometrical objects.

Molecular representations include wire models, ball-and-stick models, ribbons, space filling models, and skin representation.

### 1.4.2. Simulations

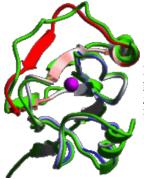
### Prediction of peptide structure from sequence



Take a peptide sequence and predict its three–dimensional structure. Of course, success is not guaranteed, especially if the peptide is longer than about 25 residues but some preliminary tests are encouraging.

You will also get a movie of your peptide folding up. Just type the peptide sequence in the \_folding file and go ahead.

### High quality models by homology



ICM has an excellent record in building accurate models by homology. The procedure will build the framework, shake up the side–chains and loops by global energy optimization. You can also color the model by local reliability to identify the potentially wrong parts of the model.

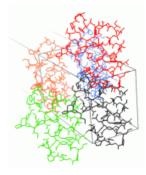
ICM also offers a fast and completely automated method to build a model by homology and extract the best fitting loops from a database of all known loops (see build model and montecarlo fast). It just takes a few seconds to build a complete model by homology with loops.

#### Loop modeling and protein design

ICM was used to design two new 7 residue loops and in both cases the designs were successful. Moreover, the predicted conformations turned out to be exactly right (accuracy of 0.5Å) after the crystallographic structures of the designed proteins were determined in Rik Wierenga's lab. Use the \_loop script to predict loop conformations and dsEnergyStrain to identify the strained parts of the design.



### Crystallographic symmetry



ICM has a full set of commands and functions to generate symmetry related molecules.

#### **Docking two proteins**

Docking two proteins reliably is still an unsolved problem. However, there has been a considerable progress. In some cases (e.g. beta lactamase and its protein inhibitor) the ICM docking procedure predicted the binding geometry correctly based only on the global energy optimization. ICM will generate a number of possible solutions using both the explicit atom model of the receptor and the receptor grid potential and refine them by explicit global optimization of the surface side–chains. Even though success is not guaranteed, the generated solutions can be useful, especially if any additional information about the binding is available.

### Finding pockets and docking a flexible ligand to a receptor

As demonstrated in several recent papers, short flexible peptides can be

successfully docked *ab initio* to their receptors. This method is a blend of the peptide folding with the grid potentials representing the receptor. A similar method can be applied to any chemical. A chemical can be built from a 2D representation and optimized. The "drugable" pockets can be predicted with an algorithm based on the contiguous grid energy densities.

### Scanning a database of flexible ligands

In virtual screening the flexible docking is applied to hundreds of thousands of individual ligands. This version of docking is fast and requires an accurate relative binding or ranking function to discriminate between the true ligands and hundreds of thousands of potential false positives. The ligand sampling and docking procedure is a combination of the genuine internal coordinate docking methodology with a sophisticated global optimization scheme.

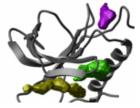
Accurate and fast potentials and empirically adjusted scoring functions have led to an efficient virtual screening methodology in which ligands are fully and continuously flexible.

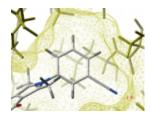
### Calculating electrostatic potential

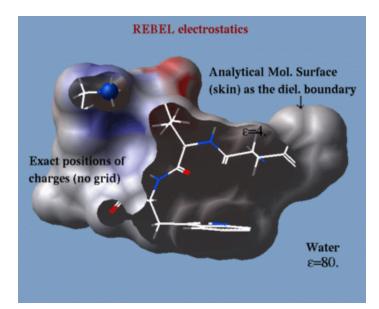
ICM incorporates a very fast and accurate boundary element solution of the Poisson equation to find the electrostatic free energy of a molecule in solution. This algorithm (abbreviated as REBEL) can be used dynamically during conformational search. The components of the electrostatic free energy are used to calculate the binding energy and evaluate the transfer energy between water and organic solvents.

ICM uses generalized Born approximation to calculate the electrostatic solvation energy and its gradient dynamically during local and global conformational searches.

The electrostatic potential can be projected on a molecular surface for the identification of possible binding sites.







# 1.4.3. Sequence analysis

#### Genomics

	Т
NM_004154 CACTTGCTAACTCTTGGATAACAAGACCTCTGCCAGAAGAACCATGGCTTTGGAAGGCGGAG	<b>T</b>
AFØØ7891G <mark>ATAACAAGACCTCTGCCAGAAGAACCAT</mark> GGCTTTGGAAGGCGGAG	TTCAGGCTGAGGAGATGGGT
NM_804154 CACTTGCTAACTCTTGGATAACAAGACCTCTGCCAGAAGAACCATGGCTTTGGAAGGGGGA AF807891	TT <mark>CA</mark> GG <mark>CTG</mark> AGG <mark>A</mark> GA <mark>T</mark> GGGT

Handling gigabytes of genomic sequence, fast cross-comparison of millions of sequences was another challenge solved in the ICM program. ICM can identify a unique subset of millions of sequences, assemble sequences from Unigene clusters into alignments (SIM4 program is used a part of the procedure).

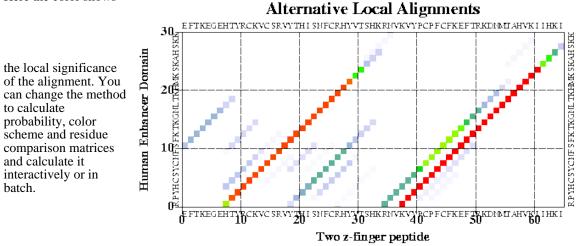
### Similarity dotplot: alternative alignments and repetitive subdomains

It looks like this:

Using the plotSeqDotMatrix macro:

```
read sequence s_icmhome + "zincFing.seq"
plotSeqDotMatrix 2drp_d 3znf_m \
"Two z-finger peptide" "Human Enhancer Domain" 5 20
```

#### Here the color shows



#### Pairwise sequence alignment and its significance

Make a pairwise sequence alignment and evaluate the probability that the two aligned sequences share the same structural fold. The alignment is performed with the Needleman and Wunsch algorithm modified to allow zero gap-end penalties (so called ZEGA alignment). The ZEGA probability is a more sensitive indicator of structural significance than the BLAST P-value. The structural statistics was derived by Abagyan and Batalov, 1997:

```
read sequence s_icmhome + "sh3.seq"
show Align(Fyn Spec) # the probability will be shown
```

You can change residue comparison matrices, gap penalties and do many alignments in batch.

#### Multiple sequence alignment

Read any number of sequences in fasta or swissprot formats and automatically align the sequences, interactively or in a batch. It will look like this:

```
# Consensus ...#.^.YD%..+~..-#~# K~-.#~###.~~..~WW.#. ~~.~G%#P.
Fyn ----VTLFVALYDYEARTEDDLSFHKGEKFQILNSSEGDWWEARSLTTGETGYIPS
Spec DETGKELVLALYDYQEKSPREVTMKKGDILTLLNSTNKDWWKVE--VNDRQGFVP-
Eps8 KTQPKKYAKSKYDFVARNSSELSM-KDDVLELILDDRRQWWKVR---NSGDGFVPN
# nID 7 Lmin 56 ID 11.5 %
#MATGAP gonnet 2.4 0.15
ICM commands:
```

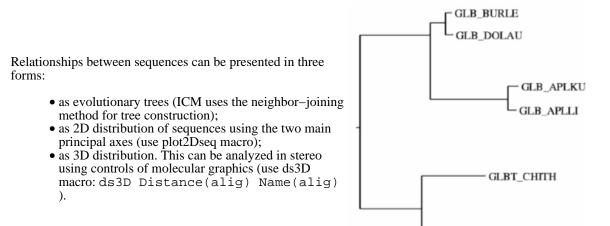
```
read sequence s_icmhome + "sh3.seq"
group sequences sh3
align sh3
```

show sh3

The gui version of ICM also has a multiple alignment viewer with dynamic coloring according to conservation tables CONSENSUS and CONSENSUSCOLOR. It will automatically show secondary structure and other features.

1IRK 96	KGFTCH-H <mark>VVRLLGV</mark> -VSKGQPTLVVMELMAHGDLKSYLR
1I44 96	KGFTCH-H <mark>VVRLLGV</mark> -VSKGQPTLVVMELMAHGDLKSYLR
1FGI 81	KMIGKHKNIINLLGA-CTQDGPLYVIVEYASKGNLREYLQ
1BYG 94	TQ <mark>L</mark> R-HSN <mark>LVQLLGV</mark> IVEEKGGLYIVTEYMAKGSLVDYLR
1QPE 104	KQLQ-HQRLVRLYAVVTQEPIYIITEYMENGSLVDFLK
	KQLQ-HQRLVRLYAVVTQEPIYIITEYMENGSLVDFLK
3LCK 104	KQLQ-HQRLVRLYAVVTQEPIYIITEYMENGSLVDFLK

#### Evolutionary trees, 2d and 3d sequence clustering



### Sensitive Sequence Similarity Search, ZEGA

Search your sequence (interactively or in batch) through any database and generate a list of possible homologues which are sorted and evaluated by probability of structural significance. The ZEGA alignment (full dynamic programming with zero end gaps) is used for each comparison and an empirical probability function described in JMB,1997 is used to assign a P–value to each hit. This search may give you more homologues that a BLAST search! The output may presented in a linked table form:

#### Table of hits

NA1	NA2	ID	SC	pР	DE
Fyn	lnyf_mNo	100.	62.81	20.94	fyn
	lines skipped				

GLBZ CHITH

```
      Eps8 ltud_m17
      21.
      17.04
      4.17
      alpha-spectrin

      Eps8 lfyn_a23
      22.6
      17.02
      4.16
      phosphotransferase fyn

      Eps8 lefn_a25
      22.
      16.64
      4.11
      fyn tyrosine kinase

      Eps8 lhsq_mNo
      24.2
      16.87
      4.1
      phospholipase c-gamma (sh3 domain)
```

### **3D plots of functions**

Take a matrix and represent it in 3D in a variety of forms. View it in stereo, color, label, transform with the mouse. Example:

```
read matrix s_icmhome + "def"
make grob def solid color
display
```

# 1.4.4. Modules of ICM

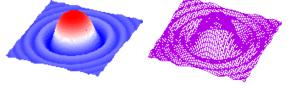
ICM is distributed in the following packages:

- ICM-main
- ICM-bioinfo (sequence analysis)
- ICM-REBEL (electrostatics)
- ICM-docking (includes cheminformatics)
- ICM–pro (includes the above four modules)
- ICM-homology (fast homology building and database loop searches in addition to ICM-pro)
- ICM–VLS (virtual ligand screening, includes ICM–pro)

The modules have the following features:

#### ICM-main

- shell for molecules, numbers, strings, vectors, matrices, tables, sequences, alignments, profiles, maps
- ICM–language and macros
- graphics, stereo
- imaging and vectorized postscript
- animation and movies
- mathematics, statistics, plotting
- presentation of the results in html format
- user-defined and automated interpretation of web links
- HTML-form-output interpretation
- pairwise and multiple sequence alignments, evolutionary trees, clustering
- secondary structure prediction and assignment, property profiles, pattern searching
- superpositions, structural alignment, Ramachandran plots
- protein quality check
- analytical molecular surface
- calculations of surface areas and volumes
- cavity analysis
- symmetry operations, access to 230 space groups



- database fragment search
- identification of common substructures in PDB
- read pdb, mol2, csd, build from sequence
- energy, solvation, MIMEL, side-chain entropies, soft van der Waals, tethers, distance and angular restraints
- local minimization
- ab initio peptide structure prediction by the Biased Probability Monte Carlo method
- loop simulations
- side-chain placement

### **ICM-REBEL** (electrostatics)

- electrostatic free energy calculated by the boundary element method
- coloring molecular surface by electrostatic potential
- binding energy (electrostatic solvation component)
- maps of electrostatic potential and its isopotential contours

#### ICM-docking and chemistry

- indexing of chemical databases in SD, mol2 and csd format
- searching and extracting from the indexed databases
- fast grid potentials
- scripts for flexible ligand docking
- scripts for protein–protein docking
- 2D (SMILES) to 3D conversion, type and charge assignment, mmff geometry optimization, low–energy rotamer generation
- refinement in full atom representation

#### **ICM**-bioinformatics

- fast comparison and redundancy removal of millions of genomic or protein sequences
- multiple EST clustering, alignment and consensus derivation
- database indexing and manipulations
- functions to evaluate sequence-structure similarity
- scripts to recognize remote similarities in the protein sequence and PDB databases
- search a pattern through a database
- searching profiles and patterns from the Prosite database through a sequence
- HTML representation of the search results with interpretation of links
- interactive editor of sequence-structure alignment
- automated building of models by homology with loop sampling and side-chain placement (fast homology model building combined with the database loop search is a separate module which is ICM Homology).

#### **ICM–Homology**

- sequence-structure alignment (threading)
- ultra-fast automated homology model building with a database loop search
- loop modeling and refinement, side-chain placement
- surface analysis

As a method for structure prediction, ICM offers a new efficient way of global energy optimization and versatile modeling operations on arbitrarily fixed multimolecular systems. It is aimed at predicting large structural rearrangements in biopolymers. The ICM–method uses a generalized description of biomolecular structures in which bond lengths, bond angles, torsion and phase angles are considered as independent variables. Any subset of those variables can be fixed. Rigid bodies formed after exclusion of some variables (i.e. all bond lengths, bond angles and phase angles, or all the variables in a protein domain, etc.) can be treated efficiently in energy calculations, since no interactions within a rigid body are calculated. Analytical energy derivatives are calculated to allow fast local minimization. To allow large scale conformational sampling and powerful molecular manipulations ICM employs a family of new **global optimization** techniques such as: Biased Probability Monte Carlo (Abagyan and Totrov, 1994), pseudo–Brownian docking method (Abagyan, Totrov and Kuznetsov, 1994) and local deformation loop movements Abagyan and Mazur, 1989).

A set of ECEPP/3 energy terms is complemented with the solvation energy, electrostatic polarization energy and side-chain entropic effects ( Abagyan and Totrov, 1994), making the total calculated energy a more realistic approximation of the true free energy. The MMFF94 force field has also been implemented. Powerful molecular graphics, the ICM-command language, and a set of structure manipulation tools and penalty functions (such as multidimensional variable restraints, tethers, distance restraints) allow the user to address a wide variety of problems concerning biomolecular structures.

# 1.5. Notational conventions

The following notational and typographical conventions are used throughout the manual.

- **Bold.** Command names may appear in bold in syntax descriptions. (e.g. **montecarlo**). Type them as they appear in the text.
- Typewriter font is used for command words, examples and ICM-shell prompts. This text can also be copied into the shell.
- *Italic* font is used for command or function arguments which should be replaced with actual values. For example, if you see

/whatever/your/ICM/directory/

and your ICM directory actually is

/usr/pub/icm

the latter is what you should actually type.

Short prefixes shown in parentheses may be used to specify argument type: integer (i), real (r), string (s), logical (l), preference (p), iarray (l), rarray (R), sarray (S), matrix (M), sequence (seq), profile (prf), alignment (ali), map (m), graphics object, or grob (g), structure factor (sf), atom selection (as), residue selection (rs), molecule selection (ms), object selection (os), variable selection, e.g. a subset of torsion angles, (vs), and table (T). These prefixes are also used to construct formal argument names for macros. For example,  $I_Color$  would mean an integer array with color information, or  $s_ObjName$  would mean a string variable or constant (e.g. "crn") specifying the object name.

• Optional arguments appear in square brackets [].

• Braces { } are used for mutually exclusive groups or arguments. For example:

set charge as\_ { r\_Charge | add r\_Increment } means either

set charge *as\_r\_Charge* or

set charge *as\_* add *r\_Increment* 

• The default values in ICM macros are shown in parenthesis and in typewriter font:

icmPocketFinder as\_receptorMol r\_threshold (3.) l\_display (yes)

# 1.6. Common abbreviations

In addition to the abbreviated ICM-shell-objects prefixes (see above), abbreviations may be used for energy terms, and some other frequently used words.

abbr.	description
as_	atom selection
ali	alignment
conf	conformation
cn	distance restraints
grad	gradient
ey	energy
hb	hydrogen bonds
ls	list
ms_	molecular selection
MC, mc	montecarlo
MB	Mouse Button
mn	maximal number of <i>items</i>
n	number of <i>items</i>
os_	object selection
re, res	residue
rs_	residue selection
rs	variable restraint
seq	sequence
to	torsion
tz	tether
ty	type
va, var	variable internal coordinate in a molecule (torsions, phase angles, planar angles, bond lengths).
VW	van der Waals
wt	weight

It is convenient to declare these abbreviations as aliases to the corresponding full words in the \_startup file for fast typing. For example:

ls seq

instead of

list sequence

# 1.7. Getting started

Start the GUI (Graphics User Interface) version of ICM by typing icm -g or icm -G and hitting RETURN. This executable will look the \$ICMHOME shell variable. The commands of the GUI menu will be taken from \$ICMHOME/icm.gui file. Feel free to change it. The GUI is meant to be self-explanatory. In this manual we will mostly focus on the shell commands and function, since in many cases the GUI gives you only limited subset of possibilities.

## 1.7.1. ICM-shell

ICM–shell is a basic interface between a user and the ICM–program. The shell can be used from the GUI version or directly. This is a powerful and flexible environment for a multitude of versatile tasks ranging from mathematics and statistics to very specialized molecular modeling tasks.

Start ICM by typing:

icm

Make sure that your .cshrc login file contains

```
setenv ICMHOME /whatever/your/ICM/directory/is/
```

Do not forget the slash at the end. It is also useful to add your \$ICMHOME directory to your \$path since there are some ICM related shell scripts and utilities which you may want to access.

You will see the ICM–prompt inviting you to type a command. The first thing to know is **how to get help**. You may just type **help** and use *l* whatever to find what you want, or use **help commands** or **help functions** to find out about the syntax. Now type:

aa=2.4

You have just created a new ICM-shell variable aa and assigned a value of 2.4 to it. You can create a variable with a name which is not already in use in the ICM-shell, does not contain space or delimiters like ".","," and starts from a letter (e.g. 1aag is an illegal name, except for sequences). Let us go on:

bb=2.\*aa

Now you have created another ICM-shell variable bb and its value is probably 4.8. Find it out by typing:

print " bb=", bb

or any of these commands:

list "b\*" list integers show bb

The next step would be to type a conditional expression like:

if (bb != 4.8) print "something went wrong"

or something even more elaborate:

```
if (bb != 4.8) then
    print "something went wrong"
else
    print "It really works"
endif
```

You can always start a for-loop such as:

```
cc={"sushi","sashimi","negi maki","toro","period."}
for i=1,Nof(cc)
# Nof returns the number of elements.
# Index i runs from 1 to 5
    print "*** I just like to eat ",cc[i]
endfor
```

Notice that anything after a pound sign # in ICM scripts is a comment.

We have just played with a real variable bb and string array cc. They had their unique names and we could create, read, write, delete and rename them.

#### **ICM-shell objects**

Furthermore, the ICM-shell can handle many other different types too, namely, it may contain in its memory entities of 16 different types, such as

```
• integer, (e.g. a=10, b= -3)
```

```
• real, (e.g. c = -3.14)
```

```
• string, (e.g. d = "ICM rules")
```

```
•logical,(e.g. e = (2 > 43); f = yes)
```

- preference, (i.e. fixed multiple choices, try show wireStyle)
- iarray, (i.e. integer arrays, g={-2,3,-1})
- rarray, (i.e. real arrays, h={ -2.3, 3.12, -1.})
- sarray, (i.e. string arrays, i={ "mek", "yerku", "erek" } )
- matrix, (read from a disk file, e.g. read matrix "def.mat")
- sequence, (i.e. amino acid or nucleotide sequences)
- alignment, (i.e. pairwise or multiple sequence alignments, read from a file)
- profile, (i.e. protein sequence profiles)
- map, (i.e. density functions defined on the 3D grid)
- grob (abbreviation for GRaphic OBject, which is different from molecular graphics objects, and contains dots, lines and solid surfaces; it can be a contoured electron density, 3D plot, an arrow,

etc)

- atomic/molecular objects and related selections of atoms (a\_//ca,c,n) residues (a\_/2:15) molecules (a\_1.b,c/) objects (a\_1,2.) and, finally ...,
- table, or spreadsheet. Several arrays are linked together in a table. Table can also have a header with some additional data fields. Tables are essentially simple databases which can be manipulated with, sorted and searched with ICM commands.

The more complicated objects, like arrays, sequences, alignments, maps etc., can be read from a disk file (e.g. read sequence "a.seq") or created by an ICM command or function (e.g. a=Sequence("ACFASDTRSEEDFFF") or make sequence a\_1.1)

Atomic objects are usually specified by an atom, residue, molecule or object selection which are collectively referred to as selections.

All of the listed entities have their unique names in the ICM-shell and can be read, renamed (e.g. rename myFactors bbb), deleted (e.g. delete myFactors aaa), written to a file with a standard type-specific extension (e.g. write aaa "surf" will create file surf.gro, the extension type depends on the object), shown, often printed and displayed graphically.

A number of ICM-variables have reserved names and are used by the program. For example, the mncalls variable always describes the number of molecular energy evaluations during a minimization, s\_pdbDir is the path to your pdb files, etc. You may customize some of those ICM-shell variables by redefining them in the system-wide \_startup file, and \$HOME/.icm/user\_startup.icm file. The standard \_startup file reads icm.ini file which contains many standard directory and parameter definitions, e.g.:

read all s\_icmhome+"icm.ini" # initialize icm variables

**Important:** be careful when negative numbers appear in the command line. If not separated from the previous numeric argument by a **comma**, they will be interpreted by ICM–shell as an expression, i.e. the two arguments will simply be replaced by their difference. For example, the command

```
display string "I like crambin" -0.9 -0.3
```

is **wrong**, a comma is needed, otherwise -0.9 -0.3 will be substituted by -1.2. This command will place the string in a point with screen coordinates x=-1.2 and y=0.0 (the default), not in x=-0.9 and y=-0.3 as might be expected. The safest way should be to use commas as separators in the argument list in the command line, like the following:

display string "I like crambin" -0.9 , -0.3

is correct, the two arguments are separated by comma

Now you can use the mouse to rotate and translate molecules and strings. The left mouse button is associated with rotation, the middle mouse button is translation and the right mouse button clicks are used for drop down menus in GUI and labeling (double click is a residue label). A more detailed list of graphics controls is given below.

As far as the keyboard commands and prompting, try to use the arrow keys for invoking previous commands and TAB for prompting (e.g. atom **TAB**) to see the available commands and functions.

## 1.7.2. The first steps

You first ICM commands may be the following:

```
read pdb "lcrn"
display ribbon
```

or simply

nice "lest"

You can also:

read mol s\_icmhome + "ex\_mol" # or read mol2 s\_icmhome + "ex\_mol2" # or read csd ...

The second way to create a molecular object is building the extended chain given the amino-acid sequence. The simplest way to build a short peptide is to use alias BS to the build string command. Type

BS ala his leu tyr # or buildpep "AHLY;AGGAR" # to build two molecules

In a more complex case create a file, say mymol.se, .se being the standard extension for the object sequence files. The file should contain the names of molecules (field ml) and their sequence (field se) and may look like this:

ml mol1
se nter ala gly his ser trp cooh
ml mol2
se hoh
ml mol3
se hoh

Type:

build "mymol"

to build the object. Now you can display the three molecular objects you have just loaded, i.e. crambin, the two peptides. We will use the cpk and the xstick graphics representations.

```
display a_2.  # a_2. means 'the second object'
display cpk a_1./2:10  # a_.. means 'residues 2:10 of the first object'
display xstick a_1./16:18
```

You can also replace residues with the modify command:

modify a\_2./his "tyr"

Let us clear the scene and start doing some more fun things:

delete a\_\*. # a\_\*. selects all the objects
build "mymol"

1.7.2. The first steps

Of course, there is a more elaborate possible setup for a montecarlo run (see \_folding script) and graphics should not be used for a real run. However, the above example is pretty much what you need to do to run the Biased Probability Monte Carlo Minimization to find the global minimum which models the solution structure of this peptide.

Now let us make a quick tour into multiple sequence alignments. First, get your sequence file (most formats will be accepted). The simplest default file format (then you do not need format type specs like: msf, pir, etc) is the fasta format (angular bracket and sequence name followed by the sequence)

> seq1
ASDFREWWDYIEQ
> seq2
SDRTYIEQWWDCVN

There are some example multiple sequence files in the ICM-directory. Let us do the following:

If you want to go directly to more elaborate sessions and scripts, or have a "How can I ..." question, you may hop to the User's guide section.

# 2. Reference Guide

# 2.1. ICM command line options

Option	Description
-a arg_string	initialize s_icmargs string with the arg_string
-b	inhibit Buffered output
-c	clean: do not save _seslog
–e 'commands'	execute semi-colon separated icm commands and quit
-g [menuFile]	start GUI menu bar, using menu file [default=icm.gui]
-G [menuFile]	start GUI menu and keep the original terminal window
-n	do Not execute _startup file
—s	Silent mode (l_warn=no l_commands=no l_info=no l_confirm=no)
-р	set path for ICMHOME, e.gp/opt/icm/
-w	web cgi mode: combination of -p and -s, e.gw/opt/icm/
-d(or -display) address	sets/redirects the X display (default is \$DISPLAY)
-24	enforce high quality 24-bit image mode at the expense of double buffering
-B[max_beeps]	no more than max_beeps on errors (default=300)
-X	report the computer identification number for a node–locked license GUI options:
-style={motif windows platinum cde}	sets the GUI style
-session=session	restores the earlier session
-geometry WxH+X+Y	sets the client geometry of the main widget, e.g. –geomerty 200x200+150+700
-fn or -font font	defines the GUI font
-bg or -background color	sets the default background color
-fg or -foreground color	sets the default foreground color
-btn or -button color	sets the default button color
-name name	sets the GUI name
-title title	sets the title (caption)
-visual TrueColor	forces to use a TrueColor visual on an 8-bit display
-ncols count	limits the number of colors on a 8-bit display
-cmap	causes to install a private color map on an 8-bit display

# 2.2. Command line editing

(cursor is in the text window).

#### Operation

Shortcut Key

command word completion/prompting TAB

2. Reference Guide

up-history	<b>UP</b> arrow
down-history	<b>DOWN</b> arrow
forward–char	<b>RIGHT</b> arrow
backward–char	LEFT arrow
beginning-of-line	<b>CTRL</b> +A
delete-char	<b>CTRL</b> +D
end-of-line	<b>CTRL</b> +E
backward-delete-char	Backspase or CTRL F+H
kill-to-line-end	<b>CTRL</b> +K
insert-overstrike toggle	CTRL+O
paste	<b>CTRL</b> +P
delete/copy-whole-line	<b>CTRL</b> +U
delete/copy-word	CTRL+W
yank (identical to paste)	<b>CTRL</b> +Y

Use the **TAB** key when you do not know what to do or to **avoid unnecessary typing** as well as probable typos in long names. This prompting is very convenient and is consistent with the **tcsh** UNIX shell. It will not only prompt you for possible completions, but also prompt you for available files in the read command (hit TAB after the double quote mark) and available selection of items in preference.

Examples:

show Ic TAB # completes function name IcmSequence()
read pdb "TAB # gives you all local \*.pdb \*.brk files
read sequence "1a TAB # lists 1a\*.seq files

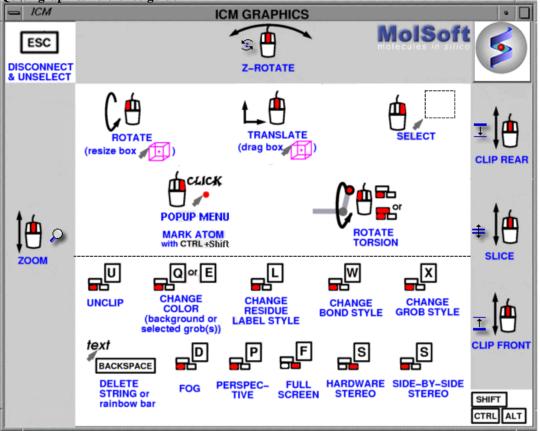
# 2.3. Graphics controls

The rough picture is simple: rotate with the *left* mouse button, translate, drag, crop, and zoom (drag along the left window margin) with the *middle* button, and select/pop with the *right* button. However these are only the defaults which can be customized.

The default shortcut keys are stored in the icm.clr file. Thefore, the mapping of keys/mouse buttons to particular graphics operations is *flexible* and can be easily redefined. The GUI controls and the **popup** menu are additional to the older shortcut keys listed here. The following shortcut keys to speed up operations in the graphics window (see the quick graphics reference guide) are defined by default. If some of these definitions are not working, check your icm.clr file in the \$ICMHOME directory and modify the key/mouse–operation mapping to your liking.

#### Quick graphics reference guide

2.3. Graphics controls



It is necessary to have the cursor in the graphics window. For some operations you may need to move cursor in a specified area (e.g. left margin) of the window. (Note for Windows 95/Windows NT version's users: if you use a two-button mouse hold the left button and the SPACE key instead of the middle mouse button (see picture-prompt for two-button mouse). Some controls use only a margin on the screen (e.g. Bottom5 means the bottom 5% of the graphics viewing area).

Note: if your SGI hardware stereo does not work properly you may need to install IRIX6.4 patches 2448, 2771 and 2843.

OPERATION	DESCRIPTION	KEYS
ROTATE	simple	LeftMB (MB stands for Mouse Button)
	continuous	Shift–Bottom5– LeftMB
SHIFT key enforces global rotation	Z-axis clockwise	LeftMB at top margin ( or ALT +Z)

TRANSLATE	Z-axis counterclockwise individual torsion angle in ICM-object XY-plane (dragging) drag atom in non-ICM object	LeftMB at top margin <i>or</i> <b>CTRL</b> +Z <b>CTRL</b> (or <b>CTRL+SHIFT</b> ) LeftMB on reference atom MiddleMB <b>CTRL</b> LeftMB at the dragged atom
SHIFT key enforces global translation	diag atom in non Terri object	
GRAPHICS.resLabelDrag controls residue label dragging	Z–axis	MiddleMB at right margin
ZOOM	zoom in	MiddleMB at left margin or <b>SHIFT</b> MiddleMB up
	zoom out	MiddleMB at left margin or <b>SHIFT</b> MiddleMB down
	front plane	CTRL MiddleMB
CLIPPING PLANES	back plane	ALT MiddleMB or Right5–MiddleMB
	slice/slab (move both planes)	CTRL+ALT MiddleMB
	unclip	CTRL+U
	label atom or grob	RightMB-click
	label residue	double RightMB-click
LABELING	paste atom's/grob's name to command line	<b>CTRL–SHIFT</b> RightMB (or under Gui: RightMB on atom and release on 1st item)
	paste residue name to the command line	<b>CTRL</b> double–RightMB (GUI: RightMB on residue, popup menu and release on 1st item. Use the residue selection level, R)
	set 3D cursor to the residue (move with arrows)	CTRL-SHIFT double RightMB
CONNECT for	disconnect/unselect everything	<b>ESC</b> or double RightMB–click, cursor in any empty area of the screen
independent movement of molecule(s) <b>SELECT</b>	connect to molecule or grob	CTRL+ALT RightMB-click on atom or vertex
<b>GROB(S)</b> for changing size or color	connect to more molecule(s)/grob(s)	CTRL+ALT+SHIFT RightMB-click
	select/edit grob add new grob to a selection side-by-side stereo on/off hardware stereo on/off full screen on/off perspective view on/off	double RightMB–click SHIFT double LeftMB–click on grob CTRL+S ALT+S CTRL+F CTRL+P

	fog ( depth cueing ) on/off	<b>CTRL</b> +D
	change resLabelStyle preference	<b>CTRL</b> +L
	change resLabelStyle preference	<b>CTRL</b> +A
	change background color	CTRL+E / CTRL+Q
	change "skin" color of the selected grob(s)	CTRL+E / CTRL+Q
	change "wire" color of the selected grobs	ALT+E / ALT+Q
	change display modes of the selected grobs	CTRL+X
MISCELLANEOUS	delete string label pointed by the cursor	BACKSPACE
MISCHEMIEOUS	gui (graphical user interface) drag the box	CTRL+G MiddleMB-click at boxCorner

# 2.4. Editing pairwise sequence-structure alignments

ICM has a simple editor for pairwise alignments. It requires that one of the sequences is linked to a structure displayed in the Graphics window. The display cursor command followed by an arrow movement displays the alignment int the top section of the graphics window. Example:

```
nice "leoc.a/"
make sequence a_1.1
read pdb sequence "3pcc.a/"
aa = Align(3pcc_a leoc_a)
display cursor a_/15 # start moving cursor with arrows
```

The *modified* alignment can be now returned by the Align(a\_1.1) command The alignment editing operations are:

OPERATION	KEYS
move cursor to the C-terminus	LeftArrow
move cursor to the N-terminus	RightArrow
move tethers to the left	CTRL+LeftArrow
move tethers to the right	CTRL+RightArrow

align 2 res./add tethers for cursor residue	INS
unalign 2 res./delete tethers for cursor res	DEL
undo editing	PAGE UP
redo editing	PAGE DOWN

# 2.5. Constants

The values of most of the ICM-shell objects may also be represented explicitly in the ICM-shell as so called "constants" (i.e. in the myFactors= $\{1.2, -4., 5.88\}$  line, myFactors is an ICM-shell variable of the rarray type, while  $\{1.2, -4., 5.88\}$  is an "rarray" constant. The following constants are defined in the ICM-shell:

- integers: -9999 12
- reals: 12.0 -0.00003 2.

V\_2//?vt\*

- logicals: yes no
- strings: "I see M", "Backslash (\\) and quote (\")" "line1\nline2"

Escape sequences which can be used inside strings:

```
\a - bell
      \b - backspace
      \f - formfeed
      n - newline
      \r - carriage return
      \t - horizontal tab
      \v - vertical tab
      \  - backslash
      \" - double quote
    • integer arrays: \{2, -1, 6, 0\} \{-8, -1, 2\} The comma is compulsory before a
      negative number, it can be skipped otherwise. See also: read iarray.
    • real array: { -1.6 , 2.150 3., -160. } See also: read rarray
    • string arrays: { "do", "re", "mi", "fa", "sol" } { "\n(newline), \t
      (tab)", "\a (bell)" } See also: read sarray
    • selections (find a detailed description below):
a_hiv?. a_1,2. a_*.
                               # objects
a_h*.a a_m1 a_*.!w2,w15,z* # molecules
```

# variables

v\_//phi,psi

# 2.6. Subsets and index expressions

one can refer to an element or a subset of ten kinds of ICM-shell variables:

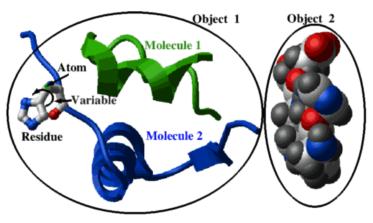
Variable type	Expression	Result type	Example
string	string[i]	string	lastChar=str[Length(str)]
	string[i1:i2]	string	resName=pdbStr[18:21]
iarray	iarray[i]	integer	CurrSize=sizes[i]
	iarray[i1:i2]	iarray	<pre>frag=list[4:nitems]</pre>
	iarray[I_]	iarray	<pre>sublist=list[{1,3,5}]</pre>
rarray	rarray[i]	real	the=same[i]
	rarray[i1:i2]	rarray	all=as[for:iarr]
	rarray[I_]	rarray	Part=R1[{1,2,3}]
sarray	sarray[i]	string	best=menu[ibest]
	sarray[i1:i2]	sarray	fish=list[4:8]
	sarray[I_]	sarray	some=all[{1,2,3}]
	matrix[i1,i2]	real	<pre>Element=M[4,5]</pre>
matrix	matrix[i1]	rarray	atomCaVec=CoordMatr[15]
	matrix[i1,i2:i3]	rarray	<pre>thirdRow=M[3,1:5]</pre>
	matrix[i1,?]	rarray	<pre>thirdRow=M[3,?]</pre>
	matrix[i1:i2,i3]	rarray	<pre>firstColumn=mm[1:3,1]</pre>
	matrix[?,i3]	rarray	<pre>firstColumn=mm[?,1]</pre>

	matrix[i1:i2,i3:i4]	matrix	upperSqr=rot[1:2,1:2]
sequence	sequence[i]	string	amino4=bpti[4]
	sequence[i1:i2]	sequence	domain=seq[139:302]
alignment	alignment[i]	alignment	column4=globins[4]
	alignment[i1,i2]	string	AminoAcIn2ndSeq=globins[4,2]
	alignment[i1:i2]	alignment	<pre>motif=EFhand[15:27]</pre>
profile	profile[i]	profile	His=prof[18:18]
	profile[i1:i2]	profile	<pre>motif=prf[14:35]</pre>
selection	selection[i]	selection	ca18=ca[18]
	selection[i1:i2]	selection	frag=ca[14:35]
table	table[i]	table	show t[3]
	table[i1:i2]	table	delete t[3:5]

**Important note**. When both lower and upper limits are explicitly specified, even if they are equal (e.g. *list* [3:3]), the type of the subset object remains the same. If only one element is specified, the rank may be reduced. The upper limit may be larger than the actual limit (e.g. t[3:9999]). You may also use 0 instead of the last element number (e.g. t[3:0]).

# 2.7. Molecule intro

Molecules are the main inhabitants of the ICM shell. The shell can contain many objects, each of which can be a soup (this expression belongs to my friend Gert Vriend) of separate molecules. Molecules, in turn contain residues and atoms. ICM can handle both **raw** objects, as they come from a PDB file or a mol-file, and a fully prepared for molecular modeling "ICM"-objects.



The non–ICM objects can be visualized, but they need to be converted into ICM–objects to perform the most interesting modeling operations. To specify the subsets of objects, molecules, residues, atoms and internal variables, you need to learn the language of molecular selections.

A quick preview of the selection language, using the picture above as an example:

```
display a_2. cpk  # object selection (the second object)
display a_1.1 ribbon green # molecule 1 from object 1
display a_1.2/his xstick # residue his12 shown as balls and sticks
color a_/1.2/12/n* xstick blue # atoms: color nitrogens in blue
```

For an in-depth description of selections, read the next section.

# 2.8. Selections

Let us imagine that we decided to compare two structures deposited in the PDB. We will read both entries in the ICM shell, and define the following levels or organization. Each entry will form an **object**, each object will contain one or several **molecules**, protein molecules will naturally contain amino acid **residues** and residues will consist of **atoms**. Now, in the superimposed command, we will need to specify, or **select**, the molecules, residues or atoms which should be superimposed. The ICM shell language has a flexible way of selecting subsets of atoms, amino-acid residues, molecules, objects, as well as torsion angles and other internal geometrical parameters of molecules. Most of the ICM commands and functions dealing with molecules, for example, display, delete, minimize, etc., will operate on an arbitrary selection. What does a selection look like? For example, selection  $a_2./2:14/c^*$  selects carbon atoms of residues from 2 to 14 of the second object. The general syntax of a selection is the following:

prefix \_ [ object(s). ] molecule(s) / residue(s) / atom(s) or variable(s)

The object section including the *dot* (e.g. lcrn.) may be omitted. In this case the selection will be performed in the current object.

There can be as many as five sections separated by \_ . / and /,

Examples:

a\_2ins.a,b/lys,arg/ca,cb,n\* # atom selection, '\*' - any string a\_2ins.a,b/2:10/n,ca,c # atom selection v\_crn./lys,arg/phi,PSI # variable selection

(Note use of PSI torsion in the last example.)

#### Storing selections in named variables.

Selections can be assigned to a variable (e.g.  $x = a_{//c^*}$ ) and can be combined in an expression by *logical and* ( & ) or *logical or* (|), e.g.  $(a_{//n^*} a_{//ca})$ .

### **Selection Types**

Three prefix types: a\_v\_ and V\_. The Prefix defines one of the three selection types:

- atoms, residues, molecules and objects ( a\_. . )
- $\bullet$  free variables ( v\_. . )
- all variables ( V\_..)

The a\_ selection is the most popular and selects **atoms**, **residues**, **molecules** or **objects**. Therefore, there are four atom selection subtypes which are abbreviated as follows:

.

abbr.	selection name	example
os_	object selection	a_ ; a_1. ; a_1crn. ; a_*
ms_	molecule selection	a_1.2 ; a_a,b ; a_*.*
rs_	residue selection	a_/3:9 ; a_/* ; a_/"GKS"
as_	atom selection	a_1.2//ca,c,n ; a_//c*
т		-1

Two additional types of selections let you select amongst the free internal coordinates or all internal coordinates (both free and fixed). These selections are widely used in commands and functions related to energy minimization and sampling:

```
abbr. selection nameexamplevs_ selection from free internal variablev_ ; v_1. ; v_1.2//x* ; v_2//?vt*Vs_ selection from all internal coordinatesV_ ; V_1. ; V_1crn.//!phi,psi,omgA selection can also be assigned to a named variable:
```

Example:

aa =  $a_//ca,c,n = the backbone$  show aa

The object and molecule sections are separated by a period, all other sections are separated by slashes. Inside each section, arguments in a list are separated by comma (,) while ranges are separated by colon ( *from:to* ).

# **Selection levels**

There are four principal levels of selection: object selection, molecular selection, residue selection and atom or variable selection. The level is defined by the "lowest" section explicitly specified in a selection (e.g.  $a_1.1/2:4$  is a residue level selection, while  $a_{//ca}$  is an atom selection). These selections are referred to as  $os_ms_rs_a$  or  $vs_$ , respectively. If selection level is not important or the level is the lowest one (atoms or variables), selections are referred to as  $as_o rvs_-$ .

The selection level of the interactive graphics selections is controlled by the GRAPHICS.selectionLevel preference. To change it from the command line, assign this variable to an appropriate level, e.g. GRAPHICS.selectionLevel="atom".

Selection levels can be changed from the GUI interface, by changing the selection level

# Examples

Examples of different selection levels (note that object and molecule names are arbitrary):

```
a_1,3. a_mod*. a_*. a_"*benz?n*". # object selections
a_3.mol1 a_zinc a_$molNum a_*.* # molecule selections
a_/3:29,as?,ala a_/* a_*./"VHC?[!W]A" # residue selections
a//h?,c* a_//T v_//phi,psi # atom or variable selections
```

For example,  $a_1$ , 3. is an object selection, and  $a_/ala$  is a residue selection.

Each section may contain a **negation** symbol ! in the beginning. It selects *all*, *but the specified*. You can only use the negation symbol in the first position of a section and the negation will always apply to the *whole* section. For example,  $a_/!ala,gly$  is right, while  $a_/ala,!gly$  is wrong.

If object section together with the separating period is skipped, selection addresses the current object rather than all objects.

## Select by number, range, name or pattern

**Matching.** Objects, molecules, residues, atoms and variables may be referred to by their names. Objects and molecules can be additionally referred to by their sequential numbers (e.g.  $a_1.2$ ). Metacharacters, such as \*? [], can also be used for pattern matching (e.g.  $v_{//?vt*}$ ).

Full syntax. A complete description of selection syntax for each level is as follows:

# 2.8.1. Object selection

( **a**\_*obj*. or just **a**\_ for the current object ):

a\_*name*. (a\_1crn., note the dot at the end)

Selection levels

a_ namePattern.	( a_1c?n. )		
a_ relNumber.	(a_2. means the second object)		
a_num1:num2.	(a_2:5. range from object 2 to object 5)		
a_	the <i>current</i> object, it is a special case.		
a_ " commentPattern ". select by pattern matching in the object comment field.			
a_ICM.	objects of ICM type ( a_!ICM. – non–ICM objects)		
Other object types (e.g. "NMR" "Fiber" "Ca-trace" etc.) can be selected or checked with the			

Other object types (e.g. "NMR", "Fiber", "Ca-trace", etc.) can be selected or checked with the Type (  $os_2$  ) function.

Example:

```
read object s_icmhome+"all"
show a_  # the current object
show a_1,2:3.
show a_s1?.
show a_"*Th[iy]o*".//!h*
```

# 2.8.2. Molecule selection

a\_obj.mol in specified object(s),
a\_mol in the current object or
a\_\*.mol in any object
by name:

 $a_s_name e.g. a_m2 \text{ or } a_1.m2$  in the current (  $a_n$  ), or the first (  $a_1$  ) object, respectively. (Note that there is **no** dot at the end )

#### by pattern

a\_*s\_namePattern* ( a\_w\* – all water molecules in the current object)

#### by number(s)

a\_number ( a\_2 , a\_3 . 2 , 4 , 7 ) - relative number of molecule(s)

#### by range(s)

a\_*num1:num2* ( a\_2:5, a\_2:5, 10:12 ) - number range

#### by special symbol for types of molecules:

a\_specialSymbol[,specialSymbol2..]

- A peptides and proteins
- H hetatm, usually ligands and water molecules
- N nucleic acids
- S sugars
- L lipids

- W water including deuterated water (dod)
- U unknown (miscellanea)

Examples:

#### Some special cases:

a\_\* # all molecules in the current object
a\_a # molecule 'a' in the current object
a\_.a # molecules 'a' in all objects
a\_\*.a # the same as a\_.a

#### selecting water molecules from pdb-files by their 'residue-field' number.

Water molecules in PDB files are numbered and the numbers are stored in the residue field. For consistency, we convert these numbers into residue numbers. At the same time the *names* of water molecules are built sequentially like this: w1, w2, w3. This way one can use both sequential numbering via molecule names and PDB–file numbering via residue numbers.

read pdb "1sri"
show a\_w12:w15 # by molecule name, sequential numbering
show a\_w\*/719:721 # by original pdb number

#### converting any selection to molecules with the Mol function

Selection of any level, e.g. atoms, residues, and objects can be converted to molecules with the Mol ( *selection* ) function. Example:

Mol(Sphere(a\_zinc a\_1,2 8.)) # Sphere returns atoms

# 2.8.3. Residue selection

With respect to objects and molecules there are the following possibilities:

a\_obj.mol/res complete specification, (e.g. a\_\*.\*/14:19 or a\_2.3/ala). a\_mol/res the current object and the specified molecules, (e.g. a\_w\*/\*) a\_/res all molecules of the current object, (e.g. a\_/23:25) Residue field specifications (for all molecules in the current object).

#### by name:

a\_/resName(e.g. a\_/his)

#### by pattern:

2.8.3. Residue selection

a/resNamePattern (e.g.  $a_/as?$  – as nor as p). A useful tip for DNA or RNA selections. Quite often bases are modified. To select A,T,G,C,U and their modifications, use  $a_/??a$  or  $a_/??t$  or  $a_/?t$  or  $a_/?t$ 

#### by residue number(s):

a\_*numChar* (a\_/3 or a\_/15A) – PDB residue number may contain additional characters.

#### by residue range(s):

a\_/numChar1:numChar2 ( a\_/4:15, 20:25 ) - reference residue number range

#### by amino acid sequence pattern:

a\_/"seqPattern" ( a\_/ "G?GTE" ) - selects the fragment with matching aminoacid sequence.

Example selecting all residues preceding prolines (the first expression selects dipeptides with the second proline, the second one excludes prolines):

show a\_/"?P" a\_/!pro\*

#### by special symbols and expressions

#### by residue type

a\_/A - residues of "Amino" type (N- and C-termini have different type) displayed residues

 $a\_/D$  – displayed residues in the ribbon representation only residues identical to their homology target residues

 $a_/I$  – if atoms of one molecular object are tethered to atoms of another object, selection  $a_/I$  shows those tethered residues (i.e. they contain tethered atoms) which have identical names to the residues to which they have been tethered.

#### by absolute number

a\_/N absNumber ( a\_/N15 ) - absolute number (all residues

of all objects are numbered sequentially starting from one.) by secondary structure

 $a_S sec_struct_chars$  – residues with certain secondary structure (e.g.  $a_SH$  – only helices;  $a_SH$  – sheets and helices;  $a_SH$  – only coil)

#### by alignment consensus

a\_/C *resConservationCode* – selects residues according to the consensus of the *alignment* linked to a molecule. The symbols can be combined, e.g. a\_/CYnh for conserved tyrosines, negatively-charged residues and hydrophobics. Possible codes:

• A, C ... – particular conserved amino acid types (one–letter code)

- X all absolutely conserved residues
- h conserved hydrophobic residues (#)
- s conserved small residues (^)
- p conserved polar residues (~)
- o conserved positive residues (+)
- n conserved negative residues (–)
- a conserved aromatic residues (%)
- x not conserved but in the ungapped block (.)
- g gap in one of the sequences of the alignment (' ')

(e.g.  $a_/CXh$  – selects all identities in the alignment and hydrophobic residues,  $a_/CACg$  – all conserved alanines, cysteins and gapped regions)

#### by functional features

a\_/**F**[*SiteChars*] or a\_/**F**"*siteID*"

residue selection by the one-letter site type or the site ID, respectively. Letter F refers to the word *feature* as in the FT (feature table) field of Swissprot entries. The types along with their one-letter codes are listed in the glossary site entry. The default string, the a\_/F selection, is defined by the SITE.defSelect string (you may redefine it), which defines important local features such as binding sites as opposed to domain-type sites such as signal peptides, zinc fingers and other protein domains. The PDB entries do not comply with the standard SWISSPROT site definitions, such as ACT\_SITE BINDING etc., and are assigned by the user type **F** (selection a\_/FF).

Example:

```
nice "las6"
show site
color ribbon a_/F magenta
show a_/FF
show a_/FF'
show a_/F"MUTAGEN" # select only site named cu3
show a_/F"MUTAGEN" # sites so defined in Swissprot
set site a_1.1 "FT SITE 15 15 My favourite residue"
```

**converting selections to residue level:** The Residue (*selection*) will convert any selection of higher level or lower level to the residue level. Example

a\_/SH a\_/pro ~# a proline in a helix Res(Sphere(a\_/pro 2.)) ~# expand to the neighboring residues

# 2.8.4. Atom selection

( **a**\_//*atoms* ):

by name

a\_//name (a\_. //ca, ca is a usual name for alpha carbon ) by name pattern

```
a_//namePattern ( a_. / / c* for all carbons )
```

#### by special symbols and expressions

#### alternative atom positions in X-ray structures

a\_//A *alterCharacter* - select alternative positions of the specified type (e.g. read pdb "lcbn" ; show a\_//Ab). See also the set comment "A" *as*\_command.

#### by atom code

 $a_{//CatomCodeNum[:atomCodeNum2]}$  – select by atom code as described in the icm.cod file, e.g.  $a_{//C2}$ , C4 selects aromatic and methylene hydrogens,  $a_{//C2}$ :15 selects codes from 2 to 15

a\_//MatomMmffCodeNum[:atomCodeMmffNum2] – by mmff code

#### displayed atoms

\*  $a_D[displayTypes]$  – Displayed atoms (e.g.  $a_//D$  for all displayed atoms, or  $a_//DWC$  for wire or cpk). The following graphical types can be selected:

- A labelled atoms
- $\bullet B ball$
- C cpk
- S skin
- W-wire
- **X** xstick (i.e. ball or stick)
- no arguments any graphical representation

#### Special named selections: as\_graph graphically selected atoms:

as\_graph selection contains graphically selected objects, molecules, residues, or atoms The level of selection depends on the GRAPHICS.selectionLevel preference. The level can be changed from the GUI interface or from command line.

#### strained atoms (atoms with high energy gradient)

 $a\_//G-strained \ atoms \ (Gradient \ vector \ longer \ than \ \texttt{selectMinGrad}) \ You \ can \ also \ use \ the \ \texttt{display} \ gradient \ command.$ 

Example:

```
buildpep "his trp trp"
display
randomize v_//phi,psi
selectMinGrad = 100.
show energy
display a_//G ball
display gradient
```

#### hydrophobic atoms a\_//H

**aromatic atoms** a\_//**R** It selects heavy atoms connected by aromatic bonds and hydrogens attached to them. Example:

buildpep "HWYP" display skin color skin a\_//R magenta

#### tethered atoms

 $a_{//T}$  – Tethered atoms (see also  $a_{//Z}$  – tether destination atoms)

#### tether-target atoms

 $a_{//Z}$  – Tether destination/target atoms (see also  $a_{//T}$  – tethered atoms)

chiral atoms  $a_{//X}[0123RLB]$  – chiral atoms. Each atom has two bits characterizing its chiral properties. If the two bits are presented as an integer, the chiral number has the following values:

- zero a non-chiral center
- 1 a left topoisomer (L)
- 2 a right topoisomer (R)
- 3 a racemic mixture of both isomers (B)

The chiral symbols can be appended. For example  $a_{//X123}$  means  $a_{//X1} | a_{//X2} | a_{//X3}$ . A short form of this selection,  $a_{//X}$  means all non-chiral atoms and is identical to  $a_{//X123}$  (or  $a_{//X0}$ ) Examples:  $a_m/3:4/X1$ ,  $a_{//XLR}$  (only left or only right chiral centers, but no racemic centers),  $a_{//XB}$  (only racemic centers)

#### by absolute number

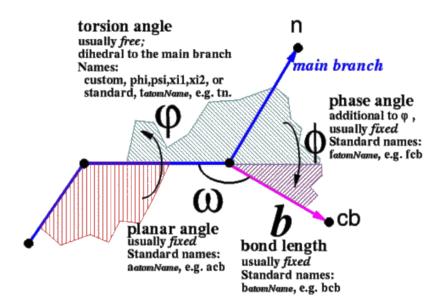
a\_//absNumber – absolute number (all atoms of all objects are numbered sequentially starting from one)

**converting to atom level:** The Atom (*selection*) will convert any selection of higher level to the atom level.

## 2.8.5. Free and all variables (v\_ and V\_)

The *v\_selection* selects **free variables** in molecular objects of ICM–type.

The main types of **internal coordinates**, or geometrical variables, are shown below:



The position of each atom branch is determined by the positions of the preceding atoms and three parameters: dihedral angle, planar angle and bond length. The dihedral angle for the main branch atom is the torsion angle itself, while for the secondary branch atoms the dihedral angle consists of the torsion angle plus the phase angle. The default fixation is given in the ICM-residue library and can be changed by fix and unfix commands. Individual free variables can be rotated interactively with Ctrl-LeftMB-Atom-Click and drag. A *vselection* can also be assigned to a named variable:

Example:

aa = v\_//phi,psi  $\ \$  the backbone torsions unfix only aa unfix only v\_/10:15/phi,psi

### V\_: selecting among all internal coordinates

Finally, the  $V_{selection}$  selects both free and fixed variables in molecular objects of ICM-type. You always need this type of selection in the unfix command. It makes no sense to unfix variables which are free already.

Here is a list of variable selection specifications:

by name:

v\_//*name*(v\_//phi)

#### by name pattern:

 $v_{//namePattern}$  ( $v_{//x^*}$ ) use asterisk \* for any string, and question mark ? for any character. Example:  $v_{//?vt^*}$  selects the 6 "virtual" variables defining rigid body rotation and translation.

#### torsion variables

 $v_{//TtorsionCodeNum[:torsionCodeNum2]}$  – select by torsion angle code as described in the icm.tot file, e.g.  $v_{//T11}$  selects the amide group torsion angle  $v_{//T10:15}$  selects torsion codes from 10 to 15

#### angles (planar angle variables)

v\_//AangleCodeNum[:angleCodeNum2] – select by planar angle code as described in the icm.bbt file.

#### bond length variables

v\_//BbondCodeNum[:bondCodeNum2] - select by bond length code as described in the icm.bst file.

#### Psi torsions not shifted to the next residue

 $v_{I}$ PSI – *psi* torsion angle which belongs to the residue you would expect. The reason for this definition is that from ICM point of the *psi* backbone torsion with rotation axis between Ca and C of residue *i* belongs to N-atom of the **next** residue *i*+1 because N is the first atom this torsion angle moves. E.g.,  $v_{J}$ /phi, psi selection will contain the *psi* from residue 2 and then *phi* from residue 3. The definition **PSI** allows you to use the conventional attribution of angles, e.g.  $v_{J}$ /phi, PSI is a pair of angles with axes around Ca atom or residue 3. **Important**. However, note that if you use selection expressions like

 $v_{\rm //phi,PSI}~a_{\rm /2,3}$  it will not work (in contrast to  $a_{\rm /2,3/phi,PSI}$  ) and you will have to use the Next function.

Example:

vPhi = v\_/3/phi vPsi = v\_/3/PSI # BUT !!! vPhi = v\_//phi\* a\_/3 vPsi = v\_//PSI Next( a\_/3 )

#### methyl group torsions

v\_//M – torsion angles rotating Methyl-type terminal hydrogens (excluding polar hydrogen)

#### polar hydrogen torsions

v\_//P - torsion angles rotating Polar hydrogens (e.g. hydroxyl group)

#### essential (non-hydrogen) torsions:

v\_//H - side chain torsion angles rotating "Heavy" atoms

#### standard set of free torsions (excludes rings)

 $v\_/\!/S$  – all "Standard" free torsion angles as defined in the <code>icm.tot</code> file.

Note that  $v_//M$ ,  $v_//P$ , and  $v_//H$  do not overlap, they are mutually exclusive.  $v_//S$  contains  $v_//M$ ,  $v_//P$ , and  $v_//H$  as well as other standard torsion angles.

## 2.8.6. Functions returning selections

- Acc select solvent-accessible atom/residues.
- Atom convert to the atom selection
- Deletion residues deleted according to the alignment
- Insertion residues inserted according the alignment
- $\bullet$  Mol convert to the molecule selection
- Next extract the next atom
- Obj convert to the object selection
- Res convert to the residue selection
- Sphere expand a selection by *r\_radius* or 5Å.
- Select selection of atoms according to their coordinates, bfactors, or other properties

**Substituting ICM-shell variables into a selection**. You can insert the value of an integer or string ICM-shell variable anywhere inside your selection by using a \$ (dollar sign) prefix. (Note, this is a general ICM-shell substitution mechanism).

Examples:

```
selstr="!w*/14:19" # a string constant
display a_$selstr
```

**Logical operations.** You can also assign selection to a variable, (i.e.:  $backbone=a_{//ca,c,n}$ ) combine several selections using logical operators (example: show  $a_{/3:6}$  backbone).

# 2.8.7. Finding contiguous residue ranges with the String function

To identify contiguous ranges of residues in residue selection, use the String (*rs*) function which will convert your selection into a string expression suitable for entering into a ICM-shell. For example, if we want to find all prolines surrounded by two other helical residues helical proline plus next and prev. residues we might do the following:

```
read pdb "ldkf"
rrange = String( a_/"?P?" ) # the result would look like "a_a.b/5:7,30:32"
rg = Split(rrange, "/, |")
                             # split into sarray with {"a_a.b", "5:7", "30:32"}
                             # bar (|) helps with multiple chains
okrg={""}
k=0 # counter for good residue triplets with HHH and ?P?
for i=2,Nof(rg)
  if Nof(Split(rg[i],":")) != 2 continue  # ignore molecular names
  if Sstructure( a_/$rq[i] ) == "HHH" then # compare with ss-pattern
   k = k+1
    okrg[k] = rg[i]
  endif
endfor
# now ok-ranges are stored in okrg string array e.g. {"5:7"}
# to use them Sum(okrg,",")
```

# 2.9. Arithmetics

Most of the ICM-objects can be used in arithmetical, logical of comparison expressions. In this section we describe operations defined in the ICM-shell.

#### Members of the arithmetic, logical and comparison expressions

Abbreviations: integer (i), real (r), string (s), logical (l), iarray (I), rarray (R), sarray (S), matrix (M), sequence (seq), profile (prf). alignment (ali), map (m), graphics object (grob) (g), atom selections (as), selections of internal coordinates, for example torsion angles, (vs), and table (T). Table arrays are abbreviated as T.I, T.R and T.S, depending on the type

# 2.9.1. Assignment

allows you to assign a value to a variable.

```
Syntax: ICM-shell-variable-name = Value or expression
```

If the name is new, a new ICM-shell variable is created.

Examples:

# 2.9.2. Arithmetic operations

The following operations are defined in the ICM-shell:

```
• addition (+):
```

- ◆ i+i returns i (e.g. 2+3 returns 5),
- ◆ i+r, r+i, r+r return r (e.g. 2+3. returns 5),
- ◆ **I**+**I** returns **I** (e.g. {1,2}+{2,3} returns {3,5}),
- ◆ I+R, R+I, R+R return R (element by element),
- ◆ s+s, s+i, s+r return s (i.e. "what" + "If" returns "whatIf", "file"+2 return "file2"),
- ◆ S+S, S+I, S+R return S (the above three operations for each element),
- ◆ M+M returns M of the same dimensions (element by element addition),
- prf+prf returns prf,
- ◆ grob + R3 return grob with coordinates translated by R3,

• map+map, map+i, map+r, i+map, r+map returns map of the same dimensions.

• subtraction ( – ) :

- ♦ i−i returns i,
- ♦ i-r, r-i, r-r return r,
- ♦ I–I returns I,
- ◆ I-R, R-I, R-R returns R (element by element),
- ♦ M–M returns M of the same dimensions (element by element subtraction),
- map-map, map-i, map-r, i-map, r-map returns map of the same dimensions.

• multiplication (\*):

- ♦ i\*i returns i,
- ♦ i\*r, r\*i, r\*r returns r,
- ◆ **I**\*I returns **I** (element by element, e.g. {1,2}\*{3,4} returns {3,8}),
- ◆ I\*R, R\*I, R\*R return R (element by element). The scalar product is returned by Sum(R\_1, R\_2), and the vector product is returned by Vector(R\_1, R\_2) (two 3D vectors)
- ◆ M\*M returns a matrix product of the two matrices (M[nk]\*M[km]==>M[nm]),
- ◆ M\*R, R\*M returns R,
- prf\*prf returns prf,
- ◆ map\*r, map\*i, i\*map, r\*map, map\*map return map (operations on each element),
- ◆ grob\*r, grob\*i, r\*grob, i\*grob return grob with transformed coordinates.
- division ( / ) :
  - ♦ i/i returns i (integer division, e.g. 3/4 returns 0),
  - ◆ i/r, r/i, r/r return r (real division, e.g. 3/4. returns 0.75),
  - ♦ I/I returns I (integer division of elements),
  - ◆ I/R, R/I, R/R return R (real division of elements),
  - ◆ map/r, map/i return map (operations on each element),

- ♦ grob/r, grob/i return grob with transformed coordinates.
- concatenation, appending into array ( // ) :
  - ◆ i//i returns I[2] (e.g. 2//3 returns {2,3}),
  - ♦ r//r returns R[2] (e.g. 2.2//3.3 returns {2.2,3.3}),
  - ♦ s//s returns S[2] (e.g. "a" / / "b" returns { "a" , "b" } ),
  - ◆ I//i returns I[n+1] extended by the integer (e.g. {1,2} / / 3 returns {1,2,3} ),
  - ◆ **I**//**I** returns **I**[**n**+**m**] (e.g. {1,2}//{3,4} returns {1,2,3,4}),
  - **R**//**r** returns **R**[**n**+1] extended by the real (e.g.  $\{1., 2.\}//3$ . returns  $\{1., 2., 3.\}$ ),
  - ◆ **R**//**R** returns **R**[**n**+**m**] (e.g. {1.,2.}//{3.,4.} returns {1.,2.,3.,4.}),
  - ♦ S//s returns S[n+1] extended by the string (e.g. { "a", "b" } // "c" returns { "a", "b", "c" } ),
  - ◆ S//S returns Sn+m (e.g. { "a", "b" } / / { "c", "d" } returns { "a", "b", "c", "d" } ),
  - ◆ M[n,m]//M[n,k] returns M[n,m+k] (matrix concatenation row by row ),
  - ◆ **seq**//**seq** returns concatenated **seq** (similar to s+s);
  - prf//prf returns concatenated prf;
  - *ali1//ali2* returns projected alignment. Projected concatenation of two alignments sharing the same sequence. The shared sequence serves as a ruler for merging the two alignments. The alignments can be of arbitrary size and number of sequences. In the simplest case of three sequences a, b, c and alignments ab and bc, the operation ab//bc will create an alignment of three sequences a b c. The function Align(ab//bc, {1,3}) will extract the, so called, projected alignment of a and c through b. Example:

	ali1	11		ali2	returns		Projected	ali.
а	VYRWA-W		b	FK-WG-	KW	а	VYR-WA	-W
b	-FKWGKW		С	AKGWAI	PGKW	b	-FK-WGF	W
						С	-AKGWAPGF	W

Additionally, character arrays (strings) can be projected from sequence to alignment and back with the String (..) function and numerical residue properties can be projected from sequence via alignment with the Rarray (..) function.

## 2.9.3. Logical operations

Logical operations with table arrays are described separately (see table in Glossary).

• and ( & ):

- ♦ 1 & I returns logical, e.g. yes no returns no
- ♦ as & as returns selection as with objects molecules residues present in both initial selections, (e.g. a\_2//ca a\_//T returns the tethered Ca atoms of the 2nd molecule),
- ◆ as & s, multiplication by a string mask, e.g. a\_//ca "x-" returns the *odd* Ca atoms.
- ◆ as & seq returns residue subselection of as with the matching sequence, e.g. a\_\*. 1crn\_m returns residues matching the crambin sequence.
- ♦ as & R returns atom subselection with coordinates within a six dimensional box array R (see also function Box), e.g.

```
read pdb "lcrn"
display ribbon
color ribbon green Res( a_//* {0.,0.,0.,9.,9.,9.} )
```

More types of selections are returned by the Select, Sphere, and Acc functions.

- vs & vs returns selection vs of internal coordinates present in both initial selections, (do not forget that v\_ are free variables, and V\_ are all variables);
- vs & as returns subset of initial variables vs which is related to selection as, e.g. side-chain torsion angles in the sphere around loop 14:18 can be selected as follows:

V\_//xi\* Sphere( a\_/14:18 )

multiplication comments on logical multiplication of two selections below.

• or (|):

- ♦ III returns I (e.g. yes | no returns yes ),
- as as returns as with members of both selections (e.g.  $a_4:6 \mid a_7/ca$ )
- vs vs returns vs with variables from both selections (e.g.  $v_//phi$ , psi |  $v_/3$ )
- vs as returns vs is equivalent to vs  $| (V_//* as)$

• not (!) :

- ♦ !I returns logical negation to the argument (e.g. !yes returns no),
- Is returns aselection of the same level with members not included in the selection argument (e.g. !a\_//ca)
- ◆ **!vs** returns **vselection** with variables not included in the selection argument (e.g. ! v\_//phi,psi )

#### 2.9.4. Comparison operators

Most of them are true comparison operators and return logical yes or no. In comparisons of table arrays or string arrays with strings, the comparison returns a subtable or subarray, respectively. Comparison operations with the table arrays are described separately (see table in Glossary).

• equal ( ==):

- ♦ i==i, i==r, r==i, r==r, s==s, I==I, R==R, S==S, M==M, as==as, vs==vs, exact equality of two objects;
- ◆ **p==i**, **p==s** return **l**. Test the value of an ICM-shell preference . Example:

```
if(wireStyle==1) print "int. or string is ok" # or
if(wireStyle=="chemistry") print "double bonds"
```

♦ S==sreturns S, a sub-sarray of elements exactly matching s (e.g. {"aa", "b", "aa", "c"}=="aa" returns {"aa", "aa"}, see also S ~ s),

• not equal ( !=) :

- ♦ i != i, i != r, r != i, r != r, s != s, I != I, R != R, S != S, M != M, as != as, vs != vs inequality of two objects;
- ◆ **p**!=**i**, **p**!=**s** return **l**. Test an ICM-shell preference . Example:

if(wireStyle != 2) print "No chemistry, sorry"

- ◆ S!=s returns S, a sub-sarray of elements not matching s (e.g. {"aa", "b", "aa", "c"}!="aa" returns {"b", "c"}, see also S !~ s).
- greater than ( > ) :
  - $\bullet i > i, i > r, r > i, r > r$
  - ◆ s > s lexicographic comparison for sorting ("apple" <"orange")
- less than ( < ) :
  - **♦** i < i, i < r, r < i, r < r, s < s
- greater or equal ( >= ) :
  - **♦** i >= i, i >= r, r >= i, r >= r, s >= s
- less or equal ( <= ) :
  - ♦ i <= i, i <= r, r <= i, r <= r, s <= s</p>
- fuzzy-equality, inclusion or pattern matching (~):
  - s~sreturns logical yes if string matches a pattern.

◆ S~sreturns S of sarray elements matching the pattern s. This comparison is similar to the UNIX grep command; it returns a subarray of lines matching the pattern rather than yes or no. Do not forget to add flanking asterisks (\*) if the pattern occurs in the middle of a string. Example:

- fuzzy-not-equal ( !~ ) :
  - ♦ s !~ s returns logical yes if string does not match a pattern s.
  - ◆ S !~ s returns S of sarray elements **not** matching the pattern s. S!~s is similar to the UNIX grep −v command; it returns a subarray of lines not matching the pattern.

#### 2.9.5. Advanced operations and some comments

- 1. Integers are automatically converted to reals in binary operations containing both integers and reals. However, in expressions like *integer1 / integer2* (the same for iarrays) they are *not* converted into reals and the result will be different from what you might expect. For example, 3/4 returns 0, but 3/4. returns 0.75.
- 2. In **s+i**, **s+r**, **S+I**, **S+R** expression numbers are automatically converted into strings. In the s+s expression the second string is simply appended to the first one.

Examples:

3. Selection arithmetics. The level of the expression  $as_1 as_2 as_3 \dots$  or  $as_1 | as_2 | \dots$  (the same with  $vs_1$ ) is defined by the **lowest level** selection in the chain ( atoms – the lowest < residues < molecules < objects ). For example, in an expression  $a_1/10 | a_2/15/cg$  the second selection is an atom–level–selection and the first one is a residue–level one. The result is the atom selection of all atoms of residue 10 plus Cg atom from residue 15.

#### 4. Selection logically multiplied by string, array, or mask

Multiplication of a selection to a string-mask or sequence. The resultant selection inherits level of the first argument. The mask is applied periodically to switch off some of the selected elements. For example mask "0001111" will switch off the first three elements in every seven. The 'switch off' characters may be the following: ' ' (space),'-','0'. Example masks to switch off the third element of five: "xx xx", "11011", "++++".

Operations upon the sequence will select only the fragment with the specified sequence from the original selection. Multiplication by an array of 6 numbers  $\{x,y,z,X,Y,Z\}$  selects atoms within the specified box.

Example:

```
read object "crn" # load crambin object
rs_ = a_/11:15 # define residue selection rs_
rs_ = rs_ "xx xx" # switch off the third element (res. 13)
display cpk a_//* {1. 0. 1. 5. 7. 6.} " # a box
```

#### 5. Transitional (or projected) alignment

Projected concatenation of two alignments sharing the same sequence. If two-sequence alignments share the same sequence, they may be merged with the shared sequence as a ruler. In the simplest case of three sequences a, b, c and alignments ab and bc, the operation ab//bc will create an alignment of three sequences a b c. The function  $Align(ab//bc, \{1,3\})$  will extract the, so called, projected alignment of **a** and **c** through **b**.

Examples of expressions:

## 2.10. Flow control

ICM contains a complete set of control statements to allow looping, jumping and conditional branching.

#### 2.10.1. Loops

Two types of loops are allowed, namely for-loop and while-loop.

For-loop

```
for <i_index> = <i_from> , <i_to> [, <i_increment> ]
    ...
endfor
```

#### While-loop

```
while( <logical_expression> )
    ...
endwhile
```

Examples:

2.10. Flow control

```
for i = 1, 9
  print "ICM-shell proudly announces that i=" i
endfor
for i = 1, 4
  print "ICM-shell proudly announces that i=" i
  for j = 1, 3
    print "ICM-shell proudly announces that nesting is possible and j=" j
  endfor
endfor
read object "crn"
for i = 1, Nof(a_/*)
                      # Nof(a_/*) means 'the number of residues'
  print Label(a_/$i)
endfor
i = -2
while (i != 4)
  i = i+1
  print i
endwhile
while(yes)
  print "endless loop, please wait 8-)"
endwhile
```

Any number of nested loops may be used.

### 2.10.2. Conditional branching

Several types of conditional statements are allowed in the ICM-shell.

#### if

```
if ( <logical_expression> ) <command>
```

#### if-then-endif

```
if ( <logical_expression> ) then
    ...
endif
```

#### if-then-elseif-..else-endif

```
if( <logical_expression> ) then
    ...
else
    ...
endif
```

#### or

```
if ( <logical_expression> ) then
   ...
elseif ( <logical_expression> ) then
```

```
elseif ( <logical_expression> ) then
   ...
else
   ...
endif
```

Note: end if or else if (instead of endif or elseif) are not accepted by ICM-shell.

Examples:

```
JohnnySaid = "The gloves didn't fit"
if ( JohnnySaid == "The gloves didn't fit" ) print "You must acquit"
grade = "bad"
if (grade == "excellent") then
  print "It's great!"
elseif (grade == "good") then
  print "It's good!"
elseif (grade == "bad") then
  print "It's not so bad!" # do not be harsh on your kids
endif
```

#### 2.10.3. Jumps

Three types of jump controls are possible, namely commands **break**, **continue** and **goto**. **break** interrupts the loop, **continue** skips commands until the nearest **endfor** or **endwhile** and continues looping, and **goto** jumps to any point below.

#### break

```
<for-loop> or <while-loop>
...
if ( <logical expression> ) break
...
<end of loop>
```

#### continue

```
<for-loop> or <while-loop>
...
if ( <logical expression> ) continue
...
<end of loop>
```

#### goto

```
if ( <logical expression> ) goto <label>
...
<label>:
...
```

#### Examples:

2.10.3. Jumps

```
for i = 1, 6
      print "currently i=", i, "and it will be increased at the next step"
      if (i == 3) then
        print "... but at this point we should stop it, sorry..."
        break
      endif
   endfor
   print "end of the loop demonstrating *break*, bye"
    for i = 1, 6
      if (i == 3) then
        print "... let us skip over step 3 and continue looping"
        continue
      endif
      print "currently i=", i, "and it will be increased at the next step"
   endfor
   print "end of the loop demonstrating *continue*, bye"
   for i = 1, 5
     if (i == 3) then
      print "... but at this point we decided to skip 3-rd step, sorry..."
       goto A
     endif
    print "currently i=", i, "and it will be increased at the next step"
    print " "
A:
   endfor
   print "end of the loop demonstrating 'goto', bye"
```

**Note:** *go to* (instead of *goto*) is not accepted by the ICM–shell. Any combination of alphanumeric characters beginning with a letter (upper or lower case) may serve as a **label.** Also keep in mind that *goto* can jump only **forward;** the **backward** *goto* is not allowed.

# 2.11. ICM molecular objects

An ICM molecular object represents one or several molecules which can coexist in physical space, so that the energy of the molecular system can be calculated. For example, if you have two homologous molecules superimposed, multiple conformations of the same structure such as NMR structure determinations or alternative positions of a side chain, they must belong to different objects. The number of objects that may be loaded in ICM is limited only by the available computer memory. Objects may be of several types (see also: the  $Type( - os_2 2)$  function):

- "ICM" the only complete type which is good for everything including energy calculations
- "X-Ray" incomplete (stripped) object created by read pdb. The structure is determined by X-ray crystallography. Good for graphics and geometrical analysis
- "NMR" incomplete (stripped) object, structure determined from NMR data, similar to the "X-ray" type above.
- "Model" incomplete (stripped) object, theoretical model also similar to the "X-ray" type above.
- "Ca-trace" incomplete (stripped) object, only alpha-carbon atoms.

• "Simplified" - simplified representation.

ICM-molecular objects are created from residues and molecules described in the ICM residue library. Its content (sequences and names of molecules) is specified in an ICM sequence file (see also IcmSequence function). An ICM-object can also be created from a non-ICM object (e.g. of X-Ray type) with the convert command.

# 2.12. Energy and Penalty Terms

The energy function calculated for any conformation of an ICM molecular object consists of individual terms described in this section. For most of them ICM calculates analytical derivatives which use gradient minimization. The terms can be switched on and off with the set terms [only] "xx, yy,..." command, e.g.

#### van der Waals ("vw")

nonbonded interatomic pairwise interactions (1–5 and further, i.e. two atoms separated by more than 3 covalent bonds). If not for tests, this terms should always be used with the "14" energy term which considers 1–4 interactions. The ECEPP/3 force field is used. Parameters are specified in the icm.vwt file and are taken from Momany et al., 1975. Both the usual 6–12 term and a soft van der Waals terms are available. See also: vwMethod, vwSoftMaxEnergy.

#### 1-4 van der Waals ("14")

A part of the total van der Waals energy for atoms separated by exactly three covalent bonds. Repulsion for 1-4 pairs is cut in half according to the ECEPP energy function. This term is complementary to the "vw" term and is usually used with the "vw" energy term.

#### Hydrogen bonding energy ("hb")

A different form of the "vw" term (10–12 instead of 6–12 for "vw") for hydrogen bonding donors and acceptors as specified in icm.cod and icm.hbt files. Parameters are taken from Momany et al., 1975. The electrostatic contribution to a given hydrogen bond is not included in "hb" and is calculated as part of the electrostatic energy.

The cutoff distance for hydrogen bonding interactions is controlled by the hbCutoff parameter.

#### Torsion energy ("to")

dihedral angle deformation energy  $K^*(1+-\cos(n^*Phi))$ . The parameters K, sign and n are given in icm.tot file. Parameters are taken from Momany et al., 1975,

**Electrostatic energy** ("el") This term is calculated in four different ways depending on the value of electroMethod preference. If electroMethod="boundary element" the solvation component is in r\_out and the envelope surface area in r\_2out.

A special case: if the van der Waals energy is calculated with the vwMethod ="soft", the electrostatic energy will be automatically buffered to avoid singularities. You will see that the electrostatic term "el" changes upon switching from vwMethod=1 to vwMethod=2. The buffering artifically increases the distance between two charged atoms to avoid having negative energy values better than the van der Waals repulsion and, therefore, will prevents collapse of oppositely charged atoms.

- 1. A simple electrostatic energy (electroMethod="Coulomb"). The Coulomb law is used to evaluate the energy. The dielectric constant is constant.
- 2. the distance dependent electrostatics ( electroMethod="distance dependent" ; currentDielConst = dielConst \* DISTANCEij ) Advantage: this term has analytical derivatives and can be used in local energy minization.
- 3. A better electrostatic free energy (electroMethod="MIMEL"), uses the Modified IMage Electrostatics approximation (Abagyan and Totrov, 1994) to evaluate both the internal Coulombic energy and electrostatic polarization free energy. Disadvantage: this term has no analytical derivatives and has no effect on local energy minimization. It can be a part of the energy function in global optimization such as montecarlo or ssearch. The solvation component is stored separately in r\_out. REBEL provides a more accurate evaluation of the electrostatic solvation energy. For small molecules, use mimelDepth = 0.3 (default 0.5).
- 4. The most accurate electrostatic free energy: (electroMethod="boundary element") which uses so called boundary element method to solve the Poisson equation to calculated a electrostatic free energy of a protein surrounded by a continuous aqueous solution. In addition to the total energy, one can extract the two components: the electrostatic solvation energy from r\_out, and the Coulomb energy can be calculated as a difference between the total electrostatic energy and r\_out.

#### Surface term ("sf"). Map m\_ga

Surface energy is based on atomic solvent-accessible surfaces. Depending on the surfaceMethod preference this term is either a surface tension which is evaluated as a product of the total solvent accessible area by the surfaceTension parameter (currently 0.012 kcal/mole/A<sup>2</sup>) or is a product of atomic accessibilities by the atomic energy density parameters similar to those proposed by Wesson and Eisenberg (1992) (check icm.hdt file). The "sf" term is evaluated at each Monte Carlo or systematic search step, but not during local minimization (we do not calculate analytical energy derivatives).

The atomic accessible surfaces are calculated using a faster modification of the Shrake and Rupley, (1973) algorithm. This algorithm analyzes all atom neighbors for each atom and Sometimes a part of molecular system is represented with the grid energy terms ( "gc", "gh") rather than by explicit atoms. In this case the atomic accessibilities need to be corrected.

This correction can be introduced with a special map, called m\_ga which stores implicit neighbor information from the parts represented with the grid potentials. The m\_ga map is calculated with the make map potential "sf" ... command (see the make map potential command), along with other grid maps.

#### Entropic free energy term (conformational entropy of side-chains) ("en")

Configurational entropy of side-chains is evaluated on the basis of their maximal possible entropy which is read from the residue library. Note that this term is calculated at room temperature (300 K), so that the ICM-shell variable temperature does not affect the entropic contribution (see Abagyan and Totrov, 1994 for values) and solvent-accessible area of a side-chain.

#### Phase angle bending term ("af")

Harmonic term  $U^*(f1-f0)^2$ . Parameters U and f0 are taken from icm.bbt file. Sometimes referred to as *improper torsion*.

#### Bond stretching energy ("bs")

Harmonic term  $U^*(b1-b0)^2$ . Parameters U and b0 taken from icm.bst file.

**Distance restraints (''cn'')** a penalty term restraining two atoms to a certain distance range. The shape of the potential is *soft square well* with lower and upper bounds. This term may be used to determine three–dimensional structure from a set of interproton distances (NOEs) resulting from NMR experiments. There are local and global distance restraints (drestraints). Local restraints become weaker and vanish as the distance grows (similar to the van der Waals forces), while global restraints become stronger as you deviate further from the required distance range.

See also files: icm.cnt and icm.cn.

#### Disulfide bonds and covalent bridges ("ss")

a penalty term establishing the additional (extra-tree) covalent bridges. Currently there are three types of covalent bridges: disulfide bonds, peptide bonds and thioester bonds. In each case several distance constraints are imposed to enforce the correct covalent geometry. The constraints for the disulfide bonds include Sg1-Sg2, Sg1-Cb2, Sg2-Cb1, Cb1-Cb2 atom pairs. The extra CO-NH bond involves C-N, C-H, O-N and O-H constraints. Similarly, CO-SH bond involves C-S, C-H, O-C, O-H, C-C and O-H constraints. The functional form of this penalty term is identical to local distance restraints. The disulfide SS bonds are automatically formed when you load the object. The disulfide bonds may be LOCAL, i.e. when two sulfur atoms feel each other ONLY at small distances. See also: icm.cnt, disulfide bond, make disulfide bond, make peptide bond, delete disulfide bond, delete peptide bond.

#### Tethers ("tz")

Quadratic restraint  $E = tzWeight *Distance^2$  between atoms in the current object and static atoms in a different object (as opposed to distance restraints "cn" between atoms in the same object). The target value of the distance is zero. See also: read pdb, set tether, and tether.

#### Multidimensional variable restraints ("rs")

Energy associated with multidimensional ellipsoidal attraction zones (in which dimension they look like soft square wells with flat bottom) in a hyperspace of internal variables (e.g. preferred side-chain or backbone torsion angles). Vrestraints are defined in icm.rst and icm.rs files and are earmarked to be used in energy calculations (as opposed as for the BPMC) with the rse field (as opposed to rs). Use set vrestraint energy command to assign vrestraints. Described in Abagyan, Totrov and Kuznetsov, 1994 (pp. 494,495).

#### **Density correlation** ("dc")

Penalty function associated with correlation between the static map ( the current map is used by default ) and a virtual map generated from atomic positions on the fly. The densityCorrMethod preference allows you to choose between several different functional forms of this term:

#### $DC = 1 - Sum(D_i - \langle D \rangle)(A_i - \langle A \rangle)/(N * Rmsd(D) * Rmsd(A))$

and  $DC = 1 - Sum(D_i - \langle D \rangle)(A_i - \langle A \rangle)/N$ 

where D<sub>i</sub> is the map value, and A<sub>i</sub> is the density generated dynamically from atomic positions.

The term has analytical derivatives with respect to the internal coordinates and can be efficiently locally minimized. By adding this term one can combine energy minimization with the real space fitting into electron density.

A more detailed description can be found in the densityCorrMethod section.

#### Crystallographic correlation between Fobs and Fcalc ("xr")

#### van der Waals grid potential for carbon probe ("gc")

van der Waals interaction between explicit non-hydrogen atoms of an ICM object and a van der Waals potential calculated on the grid. To calculate this term one needs an ICM object and map named m\_gc which is calculated with make map potential "gc" ... The calculation also counts the number of atoms in the area with Evw>0.8 \* GRID.maxVw and stores this number in r\_2out.

#### van der Waals grid potential for hydrogen probe ("gh")

#### hydrophobic potential ("gs")

#### electrostatic grid potential ("ge")

Calculates the electrostatic potential contribution from the atoms specified in the make <code>map potential as\_</code> command. The contributions are calculated by the Coulomb formula with distance dependent–dielectric constant ( $4*D_{ij}$ )

#### hydrogen bonding grid potential ("gb")

#### Potential of mean force ( "mf" and pmf )

Note that term name is "mf", while icm keyword for some commands is pmf

The mean-force "mf" potential was designed as a generic energy term which is calculated for pairs of atoms according to their pmf-types and interatomic distances. The definitions of the pmf-types and energy-distance dependencies for each contributing pair of atom types can be loaded from a .pmf file.

The list of pmf-interacting pairs is calculated dynamically and only the pairs at smaller that vwCutoff threshold distance are considered. Note: It is important that vwCutoff = 9.5 is used in binding score

evaluation.

There is a preference called mfMethod which controls if the atoms in the same molecule can interract. By default only intermolecular pairs of atoms are considered (mfMethod = 1). Switching mfMethod to 2 (or "all") allows to include all atomic pairs regardless of which molecule they belong to in the "mf" term calculation.

Since this term is quite general one can prepare different pmf-parameter files for solving different problems. The default file icm.pmf has been derived from receptor-ligand complexes and allows pmf-scoring of docked ligands. Another file: ident.pmf was designed to specify attraction of the same atom types and allows to solve a problem of chemical superposition.

The relative weight of the pmf-term is controlled by the mfWeight parameter.

An example in which we evaluate a binding score:

```
read object "rec"
read object "anwers1"
move a_2. a_1.
vwCutoff = 9.5
mfMethod = 1
show energy "mf" a_1 a_2
e = Energy("mf")
```

An example in which flexible superposition of two molecules is performed:

```
buildpep "his ; gly trp" # two molecules
read pmf "ident.pmf"
fix v_//omg
display
montecarlo "mf"
```

See also: mfMethod, pmf-file, mfWeight.

# 2.13. Integer shell parameters.

Here is the alphabetically sorted dump of integer parameters defined in the ICM-shell. These parameters are used by various commands and functions and can be changed interactively, e.g.

```
mncallsMC= 10000
montecarlo
```

ICM-shell integer variables are the following.

## 2.13.1. autoSavePeriod

In the course of a montecarlo or ssearch procedures which may run for days, the current stack of conformations which accumulates the best energy representatives of different conformational areas is saved periodically to allow access to intermediate results of the simulations. The above parameter defines the number of stack changes after which it is saved to a disk file. Set autoSavePeriod to 1 if you want to be conservative.

If you set autoSavePeriod to 0, the stack will **not** be saved at all.

Default (10).

## 2.13.2. defSymGroup

defines a crystal space group number. To find the group name and symmetry operations use the Symgroup function. Default (0) means that the group is not defined.

Examples:

```
defSymGroup = 19 # direct assignment. You know group 19, don't you?
defSymGroup = Symgroup("P212121") # Oh, you do not! ..
defSymGroup = Symgroup("P61 2 2") # This one you do not remember for sure
```

## 2.13.3. i\_out

an integer where some commands or functions store their integer output:

- Rmsd saves the number of aligned equivalent points;
- Srmsd saves the number of aligned equivalent points;
- convert saves the number of heavy atoms missing from the pdb-template (e.g. atoms of the flexible lys side-chain are not given in the pdb-file).
- superimpose saves number of aligned equivalent points;
- set tether saves the number of tethers imposed;
- set drestraint saves the number of distance restraints imposed;
- set vrestraint saves the number of variable restraints imposed;
- make disulfide bond saves the number of imposed disulfide bonds;
- minimize saves the number of function evaluations;
- montecarlo saves the total number of function evaluations during minimization;
- show area skin saves the total number of triangles in the Connolly construction.

Default (0).

## 2.13.4. iProc

the current of process number filled by the fork command. This number is zero for the parent process.

Default (0).

# 2.13.5. maxColorPotential

local electrostatic potential in kcal/e.u.charge units at which the surface element is colored by extreme red or extreme blue. All higher values will have the same color. This absolute scaling is convenient to develop a feeling of electrostatic properties of molecular surfaces.

See also: dsRebel, Potential, make grob potential.

Example:

```
build string "se glu arg" # dipeptide
maxColorPotential = 3.
dsRebel a_ yes
maxColorPotential = 6.
dsRebel a_ yes
```

## 2.13.6. maxMemory

maximal memory size requested by the program in megabytes. It is used to read blocks of databases in the search commands. Make sure that this parameter is reasonable. If your maxMemory is larger than what your computer actually has, expect serious delays. However, usually computers can handle it by swapping memory onto disk, which can be slow.

**Recommendation:** divide your available RAM by a factor from 2 to 4. Current memory resources are reported by the chkdsk command on a PC or by the top command on a UNIX workstation. Do not forget that ICM itself will *additionally* allocate some BufferSpace specified in the icm.cfg file.

Default (10.0) Mb

## 2.13.7. minTetherWindow

maximal number of preceding torsions strictly speaking rigid bodies which are locally minimized during the chain growth procedure (the minimize tether command) to create an ICM-object with ideal geometry on the basis of a set of arbitrary atom coordinates (ofter referred to as the regularization procedure).

Default (30).

## 2.13.8. mnSolutions

this parameter limits the number of hits retained by the program after a search. It is used in several icm–search functions:

- find molecule chemical substurcture search
- find pattern find sequence pattern in sequences of mol. objects.
- find database advanced sequence similarity search
- align ms\_1 ms\_2 alternatives solutions for 3D superposition
- find profile find protein Prosite profiles in a sequence
- find prosite find protein Prosite patterns in a sequence

Default (100).

## 2.13.9. mncalls

maximal number of function calls in local minimization performed in minimize, and as a part of one step of a multistep procedure in montecarlo, ssearch, convert. The number of function evaluations required to find the local minimum varies widely depending on the terms used (i.e. the "tz" term makes minimization very slow, if structure is far from its target). If the minimum is found according to the

tolGrad criterion, the procedure will be terminated anyway.

Default (100).

See also: minimizeMethod, tolGrad, drop.

#### 2.13.10. mncallsMC

maximal number of function calls in the montecarlo command. Since the procedure performs random steps accompanied by local minimization (controlled by the mncalls parameter), the number of function evaluations for the whole procedure can be roughly evaluated as a product of mncalls and the number of MC iterations. mncallsMC should be sufficiently large to ensure convergence of the global optimization procedure and may range from 10,000 for a single side-chain, 100,000 for a 3–4 residue peptide to several million calls for 15–20 residue peptide or a large protein loop.

Default (1000). The default value is small to minimize damage of the unintentional calls of the montecarlo command.

See also: montecarlo, mncalls.

#### 2.13.11. mnconf

maximal number of conformations in the conformational stack. The stack stops growing after this number is achieved and starts replacing representative conformations with higher energy values by new conformations with superior energies, if the latter are found.

Default (50)

See also: montecarlo, ssearch.

#### 2.13.12. mnhighEnergy

maximal number of consecutive accepted trial conformations which do not change the conformational stack because their energies are higher than energies of the stack conformations. Therefore, the montecarlo procedure is walking in the high energy area and is probably wasting its time. When this threshold is reached the procedure acts according to the highEnergyAction parameter.

Default (50)

See also: mnvisits, mnreject, stack.

#### 2.13.13. mnreject

maximal number of consecutive rejections (due to the Metropolis criterion) of trial conformations generated by the montecarlo procedure. When this threshold is reached the procedure acts according to the rejectAction parameter (which usually increases the simulation temperature).

Default (10)

See also: mnvisits, mnhighEnergy.

### 2.13.14. mnvisits

maximal number of visits to the same slot of the conformational stack in the course of a montecarlo procedure. When this threshold is reached the MC procedure acts according to the visitsAction parameter. A visit is an event when a newly generated conformation finds a slot with a similar conformation in it, but the stack conformation is not replaced by the new one because it has a better energy.

The optimal *mnvisits* parameter grows with the size of the problem (it may be several hundred for a 15–20 residue peptide).

Default (50)

See also: mnreject, mnhighEnergy.

### 2.13.15. nLocalDeformVar

Number of backbone torsion angle variables (excluding omegas) which are changed simultaneously to provide local deformation. This parameter can be less than the actual number of backbone torsion angles in the loop. In other words it is OK if the loop contains more than *nLocalDeformVar* variables, however, if it contains less than *nLocalDeformVar* variables, it will not be deformed.

Default (10), minimal number (8).

See also: montecarlo local.

#### 2.13.16. nSsearchStep

number of steps per variable for ssearch. Normally the whole [-180., 180.] range is divided into nSsearchStep parts. In the local mode (i.e. the search is performed around a particular conformation) the total search range around each variable is defined by the ssearchStep parameter (30. deg. by default)

Default (3).

### 2.13.17. nProc

Number of processors used by the parallel version of the program.

Default (1). Range [1: maximum\_number\_of\_processors\_available ].

### 2.13.18. randomSeed

is a seed used by the random-number generator in the montecarlo, randomize, Random function. Helpful if you need to *reproduce exactly* a calculation which uses random number(s). If the variable has its zero default value, the random function is seeded from the current time. Otherwise, if you explicitly redefine it before, let us say, a montecarlo run, it will use the specified number. Default (0).

Examples:

randomSeed=2493059372 # this number you took from the previous run montecarlo # simulation will reproduce the previous one

## 2.13.19. segMinLength

secondary structure segments shorter than this threshold will be ignored when a simplified quantitative representation of the polypeptide fold is constructed using the assign sstructure segment command.

Default (3).

### 2.13.20. sequenceBlock

length of the contiguous sequence block in sequence output.

Default (10).

See also: sequenceLine .

#### 2.13.21. sequenceLine

maximum sequence length printed on each line. Usually sequence is additionally subdivided into smaller blocks.

The same parameter also controls the size of alignment block as saved by the write alignment command.

Example:

```
read alignment s_icmhome+"sh3"
sequenceLine=1000
write sh3 "aaa"
```

Default (60). Values  $\geq 1$ .

See also: sequenceBlock

### 2.13.22. surfaceAccuracy

accuracy level used in surface calculations (not graphics). By reducing the level, you can speed up the accessibility calculation in the show area surface command. The corresponding number of dots per sphere is the following:

- Level 1 ( 89 dots )
- Level 2 (144 dots)
- Level 3 (233 dots)

```
• Level 4 ( 377 dots )
• Level 5 ( 610 dots )
```

Default (3)

See also: show area surface, "sf" energy term.

#### 2.13.23. windowSize

number of elements used for sliding window averaging by the Smooth function.

Default (7).

## 2.14. Real shell variables

ICM-shell real variables are the following.

#### 2.14.1. addBfactor

additional B-factor which may be added to the current atomic B-values to create a smoother electron density map from a set of atoms. See also: make map

Default (0.0)

### 2.14.2. alignMinCoverage

a threshold for the ratio of the aligned residues to the shorter sequence length. All alignments shorter than *alignMinCoverage\*minLength* will not be reported by find database command.

The default value is 0.5. However the parameter can be tuned with the respect to the database and the nature of the query sequence.

- Search against the protein domain sequence database: use 0.5 or higher
- Search a multidomain sequence against long multidomain sequences: use 0.1 or lower

See also: 'alignMinMethod, find database.

Default (0.5)

## 2.14.3. alignOldStatWeight

a parameter influencing the statistical evaluation of sequence comparison significance in the find database command.

Statistical significance can be evaluated in two ways: first, *a priori*, i.e. before the database search and based only on the individual score of an alignment of interest and its *theoretical* distribution, or, second, *a posteriori*, i.e. on the fly and on the basis of all *empirically* observed scores of *other* alignments in the course of the database search.

The parameter ranges from 0. to 1. and sets how two different statistical criteria of alignment significance, a precomputed (the old one) and a run-time, should be mixed. Zero corresponds to only the run-time measure ( the *new* way) in which the significance is evaluated on the run-time statistics of the observed alignment scores, while one corresponds to the statistics evaluated before the search using the formula from Abagyan and Batalov, 1997. If the database is small then the run-time score statistics may be incomplete and alignOldStatWeight closer to 1. is a better choice. On the other hand, the run-time statistics has several principal advantages:

Precomputed statistics (1.) based on individual alignment score and length	Run-time statistics (0.) based on distribution of scores
works always	relies on database diversity
is trained only in 64 condition sets and ZEGA alignment	automatically adjusts to any set of conditions, e.g. gapFunction, or alignMinCoverage
does not reflect compositional bias	automatically reflects all seq. properties
does not reflect extra terms to the score	accounts for solvent accessibility correction (see accFunction)

The run-time statistics will fit the scores to an optimized empirical function. This function avoids the problems of the normal distribution, and certain pitfalls of a popular EVD function. The resulting P-value is a reliable estimate of the false positive rate if the database is sufficiently diverse, i.e. the fraction of sequences similar to the query is small. For example, searching a tyrosine kinase through a database of tyrosine kinases will yield incorrectly low pP-values (pP = -Log(P)).

Reliable expect-values:  $P * Nof(sequences) \le 0.1$ .

Example:

- Swissprot has N=89,000 sequences. Log N = 4.95
- Reliable pP = LogN + 3, twilight pP is from LogN + 1. to LogN + 3.

#### 2.14.4. axisLength

length (in Angstroms) of the X,Y,Z axes of the coordinate frame. The axes can be displayed by the display origin or display virtual command. The axes are marked X Y Z. Example:

```
buildpep "ala ala his his"
display
axisLength=10.
display origin
```

Default (1.5)

## 2.14.5. cnWeight

weighting factor for the interatomic distance restraints penalty term. See also:  ${\tt tzMethod}$  , drestraint and <code>Bfactor</code> .

Default (1.0)

### 2.14.6. dcWeight

weighting factor for the density correlation term "dc".

Default (1.0)

## 2.14.7. densityCutoff

The neglected fraction of the total atomic electron density in the course of calculation of the grid electron density from atomic positions. Atomic density distribution is approximated by two gaussian functions which need to be truncated for computational efficiency. See also: make map command and related operations with the electron density, penalty term "dc"

Default (0.1)

### 2.14.8. dielConst

dielectric constant of the solute used in Coulomb, distance-dependent, MIMEL, and boundary element electrostatic calculations. If electroMethod="distance dependent" the actual dielectric constant is a product of dielConst and a distance from a change.

```
See also: dielConstExtern, term "el"
```

Default (4.0)

### 2.14.9. dielConstExtern

dielectric constant of the solvent exterior used in MIMEL and boundary element electrostatic calculations.

Default (78.5)

### 2.14.10. drop

expected initial function drop in local minimization. The parameter is used to evaluate initial step size. If your function is already very close to its minimum, it is a good idea to reduce the parameter, otherwise the procedure will start with an inappropriately large step.

Default (10.0)

## 2.14.11. fogStart

relative Z–depth with respect to the front clipping plane at which fogging starts. With this parameter you can keep some area in front without any fog and than start gradually increasing the effect until the back clipping plane.

To activate **fog** use **Ctrl-D**. Clipping planes can be moved with Ctrl-MiddleMB (front plane) and MiddleMB – left 5% margin (back plane). Actually the mapping of these operations to particular keystrokes is flexible and is defined in the icm.clr file. For Linux it is useful to redefine the back-clipping plane movements to

mode 9 Right5-Mid # Move rear clipping plane

Right5 means that you use the 5% right margin of your window.

Usually the fog color is the same as the background color. You can change the fog color with the

color volume Color

command.

Default (0.3)

#### 2.14.12. gapExtension

Relative gap extension penalty used in an alignment procedure. The absolute gap penalty is calculated as a product of gapExtension and the average diagonal element of the residue comparison table

Default (0.15)

See also gapFunction, Align.

#### 2.14.13. gapOpen

Relative gap opening penalty used in an alignment procedure. The absolute gap penalty is calculated as a product of gapOpen and the average diagonal element of the residue comparison table You may vary gapOpen between 1.8 and 2.8 to analyze dependence of your alignment on this parameter. Lower pairwise similarity may require somewhat lower gapOpen parameter. A value of 2.4 (gapExtension=0.15) was shown to be optimal for structural similarity recognition with the Gonnet et. al.) matrix, while a value of 2.0 was optimal for the Blosum50) matrix (Abagyan and Batalov, 1997).

Default (2.4).

See also gapFunction, Align.

#### 2.14.14. hbCutoff

(Angstroms) cutoff radius for hydrogen bonding interactions.

Default (3.0)

## 2.14.15. lineWidth

the real width of lines used to display the wire representation of chemical bonds. See also IMAGE.lineWidth parameter which controls line thickness in molecular images generated by the write postscript command, and the PLOT.lineWidth which controls the width for the plot command.

Default (1.0)

Example:

```
build string "se nad" # NAD molecule
lineWidth = 3.
wireStyle="chemistry"
display
```

## 2.14.16. mapSigmaLevel

(in Rmsd values over the mean value). Margin value used for making graphical objects contouring the 3D density map .

Default (1.5)

### 2.14.17. mcBell

average relative size of normally distributed montecarlo step from the center of an ellipsoid surrounding the multidimensional variable restraint zone.

Example:

mcBell = 1.0	#	places o	one	stand	lard	deviat	cion	at	the	rs	border
mcBell = 2.0	#	distribu	utio	n is	two	times	broa	ader	eto	2.	

Default (1.0)

## 2.14.18. mcJump

maximum value (in degrees) of random angular distortion per variable. This local random perturbation occurs if visitsAction, highEnergy or rejectAction ICM-shell variables are set to the "random" value. Randomization is a possible action in three problematic situations in montecarlo procedure.

Default (30.0)

## 2.14.19. mcShake

amplitude [Angstrom] of Brownian type the montecarlo random move applied to a molecule when one of the 6 variables defining its relative position is picked. Usually these variables may be selected by v\_myMolecule//?vt\*selection. The center of mass of the molecule randomly moves in an xyz sphere of **mcShake** radius. The molecule is also randomly rotated around a random axis with an amplitude equal to *mcShake* divided by the *MolecularRadius*. This parameter is also used as a default amplitude for the randomize command where the six position/orientation variables are selected.

Default (2.0)

## 2.14.20. mcStep

montecarlo step size (degrees). Maximum random change of one variable. This parameter is also used as the default amplitude for the randomize command

Default (180.0)

## 2.14.21. mfWeight

the overall weighting factor for the "mf" penalty term. This term may contain any user-defined energy or penalty function depending on pairs of atom types and interatomic distances. The parameters for the term are stored in icm.pmf file.

The weighting factor will determine the "mf" term contribution with respect to the energy terms.

See also: "mf" term, cnWeight, dcWeight, rsWeight, ssWeight, tzWeight, ssWeight.

Default (1.0)

### 2.14.22. mimelDepth

The fraction of an estimated molecular radius which is taken as a radius of the probe sphere used by the MIMEL algorithm. The accessible surface of this probe sphere is used to calculate the distance between a charge and the effective dielectric boundary. Described in detail on p. 991–992 of (Abagyan and Totrov, 1994). For small molecules mimelDepth = 0.3 is recommended.

Default (0.5).

### 2.14.23. mimelMolDensity

a coefficient used to calculate the effective molecular radius from a number of atoms. Recommendation: do not touch it, unless you are an advanced user. See also the description of the MIMEL method.

Default (1.0).

### 2.14.24. r\_out

a real variable where some commands and functions (e.g. show area, show volume, superimpose, minimize tether, Corr, Axis, Align) store their output. Also, in the electrostatic calculations with the MIMEL or REBEL method, the solvation energy part of the electrostatic energy is returned in

r\_out.

Default (0.0). See also:  $r_2out$ .

#### 2.14.25. r\_2out

a real variable where some commands and functions (e.g. Axis) store their output.

Default (0.0). See also: r\_out .

#### 2.14.26. resLabelShift

is the translation towards the viewer (normal to the graphics screen) used to display a label in front of cpk's or skin's rather than bury the label under them. The recommended value is 4. See also: resLabelStyle

Default (0.0) to be used with more popular wire representation.

#### 2.14.27. rsWeight

weighting factor for the multidimensional variable restraints penalty term.

Default (1.0).

#### 2.14.28. selectMinGrad

default minimal gradient vector length for gradient atom selection ( $a_//G$ ). This parameter is also used by the montecarlo fast command, which requires a value of 1.5 for optimal performance.

Example:

```
read pdb "lfox"
convertObject a_ yes no yes no
show energy
selectMinGrad=80.
show a_//G
display
display
display a_//G cpk
```

Default (1.5).

### 2.14.29. selectSphereRadius

default sphere radius (in Angstroms) for atom selections in Sphere() function, as well as the gaussian 3D averaging radius in the color ribbon command with ribbonColorStyle="reliability".

Default (5.0).

### 2.14.30. shininess

parameter defining the shininess of solid surfaces such as cpk, ribbon, ball, stick, xstick, and skin when they are displayed. Only values in the range [0.,128.] are accepted.

Example in which we generate a high quality CPK image:

```
buildpep "ASDW"
GRAPHICS.quality = 15.
shininess = 100.
display cpk
```

Default (20.0). Range: from 0. to 128.

#### 2.14.31. ssThreshold

threshold distance between two Sg atoms of cystein residues. This distance controls the automatic formation of disulfide bonds in some commands (e.g. read pdb).

Default (2.35).

#### 2.14.32. ssWeight

weighting factor for the disulfide bridge ("ss") penalty term.

Default (1.0).

#### 2.14.33. ssearchStep

angular increment (in degrees) for variables in the systematic search ( ssearch command ) in so called "local" mode when the search is performed around the current conformation.

Default (30.0).

#### 2.14.34. surfaceTension

surface energy density in kcal/mole/ $A^2$ . The surface energy which is a product of this parameter by the total solvent accessible area will be stored in the "sf" term, if surfaceMethod preference is set to "constant tension".

Note, that if a part of the system is represented with grid potentials, one needs a special m\_ga grid map for correct calculations of the surface accessibilities.

Default (0.012)

#### 2.14.35. tempLocal

montecarlo simulation temperature for local deformation random moves. This temperature can be set higher than the normal temperature since a local deformation includes a larger number of variables and may require a higher temperature for efficient sampling. To set the same simulation temperature, specify:

tempLocal=temperature

in your script.

Default (5000.).

### 2.14.36. temperature

montecarlo simulation temperature. A new trial conformation with a higher energy than the current one is accepted with the probability of  $\exp(-(Etrial - Enew)/RT))$ . RT is 0.6 kcal/mole for T = 300 Kelvin.

The effect of temperature on the montecarlo procedure is the following:

- to find the global minimum successfully one needs a combination of persistense if a chosen place with a good sense of when to stop searching in this place and move along to the next one.
- if the temperature is too high, the acceptance ratio improves (gets higher) and wider sampling becomes easier since more high energy conformations are accepted. The downside of this is the low "persistence" (or "lack of patience") of the search procedure. Instead of spending more time in each conformational vicinity to find the real global minimum, the procedure just tries a couple of sub-optimal conformations and jumps away.
- if the temperature is too low the procedure may not cover the global conformational space of interest.

Default (300.).

### 2.14.37. tolGrad

gradient tolerance criterion for local minimization. Minimization is stopped if the gradient root-mean-square deviation from zero is less than the parameter value.

Default (0.05).

## 2.14.38. tzWeight

the overall weighting factor for the tether penalty term. You may need to increase it while minimizing a highly energetically strained molecule resulting from the initial steps of the conversion or regularization procedure. Additional atom specific weights can be introduced through atomic bfactors with tzMethod="weighted"

Default (1.0).

## 2.14.39. vicinity

maximum angular root-mean-square deviation per variable (degrees) or cartesian root-mean-square deviation per atom (Angstroms) when two structures are still considered belonging to the same conformational family in conformational stack manipulations. The type of comparison is defined by the compare command.

Examples:

compare a\_//ca,c,n # compare by Cartesian RMSD vicinity = 3.0 # conf. are similar if RMSD< 3 A compare v\_//phi,psi # compare by angular RMSD vicinity = 40.0 # conf. are similar if aRMSD < 40 deg</pre>

Default (15.0). Do not forget to set it to a lower value if Cartesian RMSD is compared.

### 2.14.40. vwCutoff

(Angstroms) cutoff radius for van der Waals interactions and Coulomb electrostatics.

Default (7.5).

### 2.14.41. vwExpand

radius of a probe sphere used to display a dotted surface of a molecule. All van der Waals radii are expanded by this value. *vwExpand=0* corresponds to the CPK surface, *vwExpand=1.4* corresponds to the water-accessible surface. Be aware of the difference between the waterRadius and vwExpand parameters: waterRadius is used in

- show energy "sf"
- show [area volume] skin
- display skin

while vwExpand is used in

• show [area volume] surface

• display surface

Default (1.4).

## 2.14.42. vwSoftMaxEnergy

Parameter defining maximal energy value of van der Waals repulsion at  $r \rightarrow 0$ . for the finite approximation van der Waals function (vwMethod = "soft"). This parameter must be greater than 0. kcal/mole.

Note that in the "soft" mode, the electrostatic energy will be automatically buffered to avoid singularities. You will see that the electrostatic term "el" changes upon switching from vwMethod=1 to vwMethod=2.

Default (7.0).

## 2.14.43. waterRadius

radius of water sphere which is used to calculate an analytical molecular surface (referred to as skin) as well as the solvent-accessible surface (centers of water spheres). Because of the complexity of skin calculations, it is not recommended that one play's with this parameter (of course, you rushed to do exactly

that). Be aware of the difference between the waterRadius and vwExpand parameters: waterRadius is used in

```
show energy "sf"show [area|volume] skindisplay skin
```

while vwExpand is used in

•display surface •show [area|volume] surface

Default (1.4).

## 2.14.44. wireBondSeparation

the distance between two parallel lines representing a chemical double bond if wireStyle = "chemistry".

Default (0.15 Angstroms).

## 2.14.45. xrWeight

the overall weighting factor for the structure factor correlation penalty term. See also: xrMethod.

Default (1.0).

# 2.15. Logical variables

ICM-shell logical variables are the following.

### 2.15.1. I\_antiAlias

if yes, invokes antialising for lines displayed in the graphics window. This feature is not supported on all the platforms.

Default (no).

## 2.15.2. I\_autoLink

if yes, tries to link molecules and alignments/sequences automatically. In case of degeneracy, i.e. identical sequences exist with different names, a molecule can be linked to two different alignments containing its sequence etc., the autolink procedure chooses the first occurrence. Use the link command to impose links explicitly, and the show link command to see them. Links can be used by the following commands and functions:

- superimpose
- Rmsd and Srmsd

2.14.44. wireBondSeparation

```
• set tether ali_ ...
```

Default (yes).

## 2.15.3. I\_bpmc

if yes, use Biased Probability Monte Carlo moves in the Monte Carlo procedure. See Abagyan and Totrov, 1994 for reference. **Important:** the probability zones are described in the icm.rst file and should be assigned to a peptide before the montecarlo command with the

set vrestraint a\_/\*

command.

Default (yes).

## 2.15.4. I\_breakRibbon

if yes, break too the ribbon if the distance between the reference atoms is larger than 9 Angstroms. Default (yes).

## 2.15.5. I\_bufferedOutput

if no, suppresses paging in the output of ICM commands. Useful in batch jobs.

Default (yes).

## 2.15.6. I\_bug

if yes, print some debug information

Default ( no ).

## 2.15.7. I\_caseSensitivity

active in most commands and functions using string comparisons.

Default (no).

## 2.15.8. I\_commands

if no, do not show commands in batch mode

Default (yes).

## 2.15.9. I\_confirm

if no, overwrite the contents of an existing file; ask permission to overwrite it otherwise.

Default (no).

## 2.15.10. I\_easyRotate

allows faster handling of images in the graphics window. If yes, then the currently displayed solid representations (e.g., ribbon, skin, cpk, etc.) are temporarily hidden if an operation like rotation or translation is undertaken. Only the wire representation remains allowing quick manipulation with the object in use. The previous type of display is restored when rotation or translation is completed. The parameter can be toggled by a keystroke if you assign the l\_easyRotate = !l\_easyRotate with the set key command.

Default ( no ).

### 2.15.11. I\_info

if yes, print info messages

Default (yes).

### 2.15.12. I\_minRedraw

if no, suppresses redrawing of a displayed structure at each minimization step. The new minimized structure will be redrawn only at the end of minimization. Useful when the graphics is slow or the structure is heavy.

## 2.15.13. I\_neutralAcids

Several commands such as read mol, read mol2, build smiles and set bond auto include automated assignment of aromatic systems as well as some resonance structures in O–C=O, O–S=O, PO3, O–N=O, and NO3. The automated conversion invoked with the l\_readMolArom variable set to yes reassigns the bonds in the group to be equivalent. For the acidic groups it leads to the *charged* form with two partial charges of -1/2 or -1/3. If you want to suppress this transformation for the CO2,SO2 and PO3 groups only set the l\_neutralAcids flag to yes. In this case the acidic groups will be kept unchanged.

Example:

```
l_neutralAcids = yes
read mol s_icmhome+"ex_mol.mol"
wireStyle=2
display only a_ # the acidic group is uncharged
build hydrogen
```

Default (no).

See also: read">readmol">read mol, read mol2, build smiles and set bond auto.

#### 2.15.14. I\_out

a logical variable similar to i\_out and r\_out.

Default (yes).

#### 2.15.15. I\_print

if yes, show print command with arguments as well as the result of its action.

Default (no).

#### 2.15.16. I\_readMolArom

if yes, automatically assigns aromatic rings and resonant structures (CO2,SO2,PO3,NO2,NO3) from patterns of single and double bonds upon reading objects, mol and mol2 files or build from smiles. The automated assignment module is also called by the set bond auto command.

If this flag is set to no, the build hydrogen command will have problems with resonant structures, such as carboxyl groups, -a hydrogen will be attached to the oxygen connected with a single bond to the carbon.

Example of a recommended best convertion procedure for chemical library files:

Default (yes).

See also: 1\_neutralAcids which allows to keep acidic groups unchanged and uncharged.

## 2.15.17. I\_showAccessibility

show the residue accessibility string assigned to a sequence generated from a three dimensional structure in the commands show sequence, show alignment, write alignment. The relative residue accessible area is expressed by an integer number in a scale from 0 to 9 (0-fully buried, 9-fully exposed). Example:

```
read pdb "lcrn"
show surface area  # calculate atomic and residue accessibilities
make sequence a_1  # generate a sequence
l_showAccessibility=yes
```

show lcrn\_m

Default (yes).

## 2.15.18. I\_showMC

display one-line info about each Monte Carlo trial conformation.

Default (yes).

## 2.15.19. I\_showMinSteps

display every step of the local minimization procedure.

Default (no).

## 2.15.20. I\_showSpecialChar

if yes, displays unprintable characters with the show string and list string commands in text format (like  $\langle a \rangle (t n)$ ). This flag does not apply to the print command.

Default ( no ).

## 2.15.21. I\_showSites

show the site string assigned to a sequence in the commands show sequence, show alignment, write alignment. The one-letter site codes are given below.

Default (yes).

## 2.15.22. I\_showSstructure

show the secondary structure string assigned to a sequence in the commands show sequence, show alignment, write alignment.

Default (no).

## 2.15.23. I\_showWater

if yes, all water molecules are shown in the output of commands such as show molecule or show a\_\*. Set it to no to skip the usually long lists of water molecules in PDB structures.

Default ( yes ).

## 2.15.24. I\_showTerms

**Obsolete.** Now you can achieve the same via s\_icmPrompt variable.

Examples:

s\_icmPrompt = "icm/%o/%e> " # equivalent to l\_showTerms=yes

#### 2.15.25. I\_warn

if yes, print warning messages. If you want to see warning messages (i.e. 1\_warn = yes), but suppress some of the messages, use the s\_skipMessages variable (e.g. s\_skipMessages = "[147][148]").

Default (yes).

## 2.15.26. I\_wrapLine

wrap long lines if yes. If no truncate long lines and add a dollar sign () to indicate that truncation has occurred.

Default (yes).

## 2.15.27. I\_writeStartObjMC

write the starting object in the montecarlo command to a file. This object will have the same fixation (set of free and fixed variables) as in your montecarlo simulation. In case the variable is set to no, the same object can be generated if you repeat the fix and unfix command as in your simulation script.

Default (yes).

### 2.15.28. I\_xrUseHydrogen

defines whether hydrogen atoms are used in calculations of crystallographic structure factors from atom coordinates (the term).

Default (yes).

# 2.16. String variables

### 2.16.1. s\_blastdbDir

return directory with Blast-formatted sequence files for ICM sequence searches. By default the directory is set to the \$BLASTDB system shell variable. The variable can also be explicitly defined in the user\_profile.icm or \_startup file. In order to start using the \$BLASTDB shell variable, delete explicit assignment of the s\_blastdbDir from your \_startup file or add

s\_blastdbDir=Getenv("BLASTDB")

to your ~/.icm/user\_startup.icm file.

The find database family of sequence/pattern search commands use the s\_blastdbDir directory.

#### 2.16.2. s\_editor

a string to invoke an external editor.

Attention!!! Always use the call to the program which starts the program in the foreground. For example: use "jot –f" rather than just "jot", since the default is running in the background.

Examples:

```
s_editor = "vi"  # good old vi, does not require a separate window
s_editor = "jot -f"  # popular SGI editor
s_editor = "xedit"  # simple and exists for X on every platform
s_editor = "notepad"  # exists for PCs
```

## 2.16.3. s\_entryDelimiter

a string which delimits entries in the database output of a table or a set of arrays, generated by the show database or write database commands. The **%i** specification at the end will be replaced by the current number of the entry and carriage return.

Default: ("#\_\_\_\_\_\_%i")

Example:

s\_entryDelimiter="//\n" # EMBL-database delimiter

### 2.16.4. s\_errorFormat

Examples:

## 2.16.5. s\_fieldDelimiter

contains characters which are considered as field delimiters by the Field and Split functions, as well as by read column and write table commands. In "Split" and "read table" one can also specify the field delimiter explicitly.

**Important.** If a character is duplicated in *s\_fieldDelimiter* (e.g. s\_fieldDelimiter="::"), then multiple occurrences of this character will be ignored. Otherwise, EMPTY fields will be created between

each pair of identical delimiter characters.

In write table s\_fieldDelimiter is honored only if is a one-letter symbol, like "," or "\t".

See also the opposite operation, merging members of string array into one string: Sum(S\_, s\_separator)

Examples:

s\_fieldDelimiter="\t" # "aaa\t\t bbb" splits into "aaa",""," bbb" s\_fieldDelimiter="\t\t" # "aaa\t\t bbb" splits into "aaa"," bbb"

Default ( "  $tt^{"}$  i.e. two blanks, two tabs, meaning skip multiple blanks or tabs). Another reasonable possibility is "  $tt^{nn"}$  which means skip blanks, tabs and carriage returns.

## 2.16.6. s\_helpEngine

path to the HTML help file browser program. If you have no HTML browser, the default setting is s\_helpEngine="icm", so you can use the simple internal ascii help-file viewer more filter ('q' - to stop, '/' to find a string, 'Enter' - next screen). If the desired help information is not found, just type help and then use '/' plus the search pattern to perform the context search in the whole help file.

Examples:

```
s_helpEngine="/usr/bin/netscape"
s_helpEngine="mozilla" # make sure you can start it in the UNIX shell
s_helpEngine="icm" # why would one need more?
```

## 2.16.7. s\_icmhome

defines the home directory of the ICM program. This directory contains all standard ICM databases, all scripts, examples, documentation, initial configuration files (later users can override them with the files stored in the s\_userDir directory.

The Linux icm-rpm package creates s\_icmhome in /usr/icm directory.

### 2.16.8. s\_inxDir

defines directory from which icm – index files for large sequence or chemical databases are stored. This variable is used by the write index command. By default s\_inxDir is set to s\_icmhome + "/data/inx/".

See also: read index, write index table, write index.

### 2.16.9. s\_icmPrompt

defines the ICM-prompt string. This string contains text and a bunch of wild cards for:

- %0 name of the current molecular **object**
- %e list of the active energy terms (see the set terms command)
- %t time spent in ICM (may be convenient for scripts)

- %T astronomical Time
- %% % character
- %# icm–command order number

Be smart, see the energy or penalty terms you are using by adding %e to the prompt string.

Examples:

Default: "icm/%o> "

### 2.16.10. s\_imageViewer

defines the command to view the image files (tiff, png, targa and rgb formats) if the display option is specified. An alternative to the default is the "xv" program. See also the write image command.

Default for SGIs ( "imgview").

### 2.16.11. s\_labelHeader

defines a prefix string for all labels. For example, when displaying CPK atoms you may move the label to the right of the atom center by

s\_labelHeader="

Default ( " " – an empty string).

### 2.16.12. s\_lib

ICM library name root. If you redefine it to say "new", ICM will start to look for the following library files: new.cod, new.bbt, new.bbs, .... etc. in the \$ICMHOME directory.

Default("icm").

### 2.16.13. s\_logDir

when you quit an icm-session, a \_seslog.icm file is automatically stored. If the s\_logDir variable is empty, it is stored to the s\_userDir + "/log/" directory. However one can redirect it to the current working directory (".") or any other directory.

The same logic applies to the \_crashlog.icm file which is created when ICM crashes.

Examples:

```
<code>s_logDir = "." # _seslog.icm</code> stored in the current working directory <code>s_logDir = "" # to the current working directory</code>
```

### 2.16.14. s\_out

a string where some commands store their string/text output. See also: printfread database read string, read table, and read unix,

Default ( "is where the string/text is stored" ).

### 2.16.15. s\_pdbDir

directory containing the PDB database of 3D structures. These files can also be easily downloaded directly from the PDB site if the variables are set as in the example below. PDB distributions can exist in several styles (all files in the same directory, or divided etc.). The style is defined by the pdbDirStyle preference.

The pdb directory also contains the derived\_data subdirectory with useful files ( pdb sequences, index files etc. )

Example:

```
s_pdbDir ="ftp://ftp.rcsb.org/pub/pdb/data/structures/divided/pdb/"
pdbDirStyle = "ab/pdblabc.ent.Z"
s_pdbDir = "/data/pdb/"
read sarray s_pdbDir+"/derived_data/index/source.idx"
source = Tolower(Trim(Field(source,1)))
for i=1,Nof(source)
    read pdb source[i]
    do some analysis
    delete a_*.
endfor
```

Default ( "/data/pdb/"). It is usually redefined in the \_startup file.

### 2.16.16. s\_projectDir

a **relative** path to the directory in which icm-projects (all the icm-objects in a session) are stored. This path is appended to the s\_userDir directory.

### 2.16.17. s\_printCommand

a command to print text or postscript files. This command is invoked if the print option is specified in the write image postscript or write postscript commands. Customize this string. Default ("lp -c").

Example:

s\_printCommand = "lp -c -d ColorPrn22"
write image postscript print # save image and print

### 2.16.18. s\_prositeDat

is a file containing the full file name of the prosite database of protein patterns. This file is not large and is distributed with ICM. If you have your own copy of prosite, redefine the variable and delete prosite.dat in the \$ICMHOME directory to avoid redundancy.

 $\label{eq:loss_loss} Default ( \ \texttt{"prosite.dat"} ). \ It is usually redefined to \ \texttt{s_icmhome+"prosite.dat"} in the \ \texttt{startup file.}$ 

### 2.16.19. s\_psViewer

a PostScript viewer used while you are in ICM session. A command to invoke is to be:

```
unix $s_psViewer </tt><i>your PostScript file name</i>
```

```
Default("/usr/opt/bin/gs -q").
```

#### 2.16.20. s\_reslib

name of the icm residue library. The file will be loaded from the \$ICMHOME directory.

Default ( "icm").

### 2.16.21. s\_skipMessages : ignore specific error messages

In ICM all error and warning messages are numbered (e.g. " Warning> [123] ... "). You may specify a set of message numbers which you want to suppress. While the messages are suppressed the error code can still be returned with the Error(number) function.

Example:

```
a = 1
if = 2
             # deliberately generate error
Error> [2073] illegal IF: wrong condition in if=2
s skipMessages = "[2073]"
if = 2
              # now no message is generated
if Error(number)==2073 quit
            # generates another error
a = yes
Error> [696] wrong assignment or name conflict
s_skipMessages = "[2073][696]"
             # hides the error message
a = yes
234*2352352532
Warning> [147] number 2352352532 is too big for an integer (>2147483647)
s_skipMessages = "[2073][696][147]" # suppress the warning
234*2352352532
```

0

```
See also: errorAction, s_errorFormat.
```

```
Default ( "[3000][3012]" just to show an example).
```

### 2.16.22. s\_tempDir

scratch directory for temporary files ( some montecarlo files will be saved there ).

```
Default("/usr/tmp/").
```

### 2.16.23. s\_translateString

a set of characters used in the ascii representation of numerical values of arrays, matrices and maps. See also the String function and the show map command.

```
Default ( ".:*0#").
```

### 2.16.24. s\_userDir

The path to the user directory containing ICM-related and ICM-generated data files.

The suggested \_startup file sets this variable to a subdirectory .icm of the user \$HOME directory ( \$USERPROFILE for Windows), but you may set it anywhere you want.

Default("\$HOME/.icm/").

### 2.16.25. s\_usrlib (obsolete)

an obsolete variable. The new mechanism to add new icm residue libraries uses the LIBRARY.res sarray. You can generate the entries using the write library command.

Default ( "usr").

### 2.16.26. s\_webEntrezLink

defines the NCBI Entrez link.

See also: webEntrezOption, Default (
 "http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=s&form=6.

### 2.16.27. s\_webViewer

an HTML browser invoked by ICM by the following commands: web dbEntryCode, web T\_, write html.

Examples:

```
s_webViewer="/usr/bin/netscape"
s_webViewer="Mozilla"
```

Default ( "netscape").

### 2.16.28. s\_xpdbDir

path to the ICM XPDB database of compact binary ICM objects which are annotated with the site information. The advantage of the XPDB database is the speed of reading and smaller size than PDB. XPDB entries are read about **80** times faster!

Here we compare the execution times for the pdb and xpdb files:

```
eos:/home/ruben/icm> time ./icm -s -e 'read object "/data/xpdb/lffk.ob"'
0.450u 0.090s 0:00.54 100.0%
eos:/home/ruben/icm> time ./icm -s -e 'read pdb "/data/pdb/ff/pdblffk.ent.Z"'
38.800u 0.430s 0:42.11 93.1%
```

# 2.17. Preferences

Preferences are multiple choices. You can show and list them. You can change a preference by assigning it to:

- the item number
- the item name
- "nextItem" string
- 0 (the same as "nextItem")

Examples:

```
resLabelStyle = 3  # 3-rd choice
resLabelStyle = "Ala 5"  # assign by string
resLabelStyle = "nextItem"  # go to the next item in the list
```

### 2.17.1. atomLabelStyle

style of atom labels invoked by clicking on an atom or the display atom label as\_command. You may display name, electric charge (q) and/or mmff atom type. Options are the following:

```
1. "cbl" <== default
2. "cbl q" (atomic charge)
3. "cbl:FC" (formal charge)
4. "cbl all" (different atomic properties)
5. "cbl mmff q"
6. "C" (chemical atom name)
7. "[C]" (chem. name on a rectangle )</pre>
```

The last two choices use periodic table convention to label atoms, and the label is positioned into the center of atom. In the latter case ([C]) a rectangle of the background color is used to highlight the label.

The standard \_aliases file contains the set key "Ctrl-A" "atomLabelStyle=\"nextItem\"\n" command, which means that you may switch between different styles interactively by pressing Ctrl-A.

Examples:

```
build string "se his"
atomLabelStyle = "[C]"
wireStyle = "chemistry"
lineWidth = 3.
display atom label wire black # press Ctrl-A
color background white
write postscript "tm" # save the results
#
atomLabelStyle = "C"
display xstick
set type mmff # press Ctrl-A again
```

### 2.17.2. alignMethod

alignment method used in the Align and Score functions and find database command (as described in Batalov and Abagyan, 1999).

- 1. "ZEGA"
- 2. "H–align" <– the best choice
- 3. "frame-H-align" # allows to align DNA sequence against protein sequence or protein sequence database

See also:

- gapFunction,
- accFunction,
- alignMinCoverage (0.5) minimal ratio of the aligned residues with respect to the shorter sequence length.

### 2.17.3. atomSingleStyle

display style of isolated atoms in the wire mode.

- 1. "tetrahedron"
- 2. "cross"
- 3. "dot"

The size of the first two representation is controlled by the GRAPHICS.ballRadius parameter and the line width (especially important for the "dot" style) is controlled by the lineWidth parameter.

### 2.17.4. dcMethod

defines the algorithm for the density correlation calculation which is the correlation between the static density distribution and a virtual map generated from atomic positions on the fly.

1. "exact" <- default</pre>

```
2. "unnormalized"
```

#### Explanation:

1. The "exact" density correlation penalty function uses the Pearson's correlation coefficient. The correlation coefficient is then shifted by +1 so that the function ranges from 0. to 2. rather than from 1. to -1.

 $DC = 1 - Sum(D_i - \langle D \rangle)(A_i - \langle A \rangle)/(N * Rmsd(D) * Rmsd(A))$ 

The term has analytical derivatives with respect to the internal coordinates and can be efficiently locally minimized. This term requires additional memory allocation equal to the current map size and is two times slower than the unnormalized term.

2. The "unnormalized" density correlation. Formula:

 $DC = 1 - Sum(D_i - <D>)(A_i - <A>)/N$ 

where  $D_i$  is a map value in point *i*, and  $A_i$  represents the density generated dynamically from atomic positions.

The differences from the "exact" term are the following:

- scaling is arbitrary in contrast to "exact" term. Therefore you have to estimate a reasonable dcWeight value if "dc" is optimized along with the other energy or penalty terms.
- The "unnormalized" term does not require additional memory and is two times faster than the "exact" term. The term has analytical derivatives with respect to the internal coordinates and can be efficiently locally minimized.

# 2.17.5. electroMethod

defines method used for the electrostatic energy evaluation. Four options are available:

- 1. "Coulomb"
- 2. "distance dependent" <- default
- 3. "MIMEL"
- 4. "boundary element"

#### The meaning:

- 1. The Coulomb electrostatics is defined as  $U = q_1 * q_2 / D * r_{12}$  with D = dielConst.
- 2. In the distance-dependent dielectric model D in the above formula is set to dielConst\*r, where r is an interatomic distance.
- 3. The "MIMEL" electrostatics allows to evaluate the free energy of a molecule in water environment by the Modified IMage Electrostatics approximation at every iteration of the Monte Carlo, or search procedure. This energy will only be calculated for a static structure or at the end of local minimization ( so called "double energy scheme", see Abagyan and

Totrov, 1994 section (e) on p.992, or Abagyan, Totrov and Kuznetsov, 1994 p. 10, for reference). ). The MIMEL energy consists of the Coulomb energy, which is calculated for all the atom pairs at the current dielConst value, and the electrostatic solvation energy which is a sum of "selfEnergy" and "crossEnergy" and is returned in the r\_out real variable upon completion of the calculation in the show energy command. A more accurate evaluation of the electrostatic solvation energy can be obtained with the boundary element method.

4. The boundary element method provides an accurate solution of the Poisson equation. The dielectric boundary is defined by the accurate analytical molecular surface (skin) and all the local charges stay exactly where they are. The boundary element method does not rely on any 3D grid and is free from dependence on the grid size. The ICM implementation of the boundary element method is fast and accurate. During the local minimization the derivatives with respect to the internal coordinates are not calculated (similar to the MIMEL method). The distance dependent dielectric model is used during minimization instead. At the end of the local minimization the electrostatic energy is replaced by the more rigorous boundary element energy.

### 2.17.6. errorAction

action taken after an error has occurred.

1. = "none" # error flag is set (see the Error() function)

2. = "break" <- default # exit from loops and macros

3. = "exit" # exit from a script into shell

4. = "quit" # quit ICM: useful for CGIs

Specific error messages can be suppressed with the s\_skipMessages ( e.g. s\_skipMessages =
"[696][2073]" )

See also: s\_errorFormat, interruptAction

### 2.17.7. ffMethod

force field used in the show energy, minimize, and montecarlo commands.

```
1. = "ecepp" <- default
```

```
2. = "mmff"
```

3. = "icmff" a new experimental force field obtained by reparametrization of the mmff force field into the internal coordinate space and derivation of the parameters specific for a particular covalent geometry.

Note that minimize cartesian temporarily enforces ffMethod = "mmff", since the ecepp force field is not applicable to the cartesian minimization.

To use the force fields you need to do the following:

- "ecepp"
  - read library (if it is not included in your \_startup.icm file)
  - ♦ modify terms with the set terms command.
  - ♦ use show energy, minimize, or montecarlo.

- "mmff" in cartesian space (free covalent geometry). The command requires at least the
  - "vw, af, bb, bs" terms and needs correct atom types and charges.
    - + read library mmff
    - ♦ assign atom types: set type mmff a\_. This operation requires correct
- chemical structure (when you build the molecule, make sure it is complete),
- bond types (check graphically with wireMethod=2, and change with the set bond type command), and
- formal charges (check graphically with the atomLabelStyle=3, and assign with the set charge formal .. command).
  - ♦ assign charges: set charge mmff a\_
  - modify terms with the set terms command. The full set is: set terms
     "vw,el,to,af,bb,bs"
  - ♦ use show energy, minimize, or montecarlo.
- "mmff" in the internal coordinate space according to the current fixation. The use of the mmff force field is not recommended.
- "icmff". This new force field is designed to be used with the fixed covalent geometry and is faster than both mmff-cartesian and "ecepp". The icmff force field is still experimental and should be used with caution. The vacuum part of icmff requires only three terms: "vw,to,el". The solvation terms "sf,en" can be added.

Icmff calculates parameters on the fly for a particular geometry. To use this force field use the following procedures:

- assign mmff types and charges, and load the mmff libraries (see above)
- to generate the starting conformation, minimize your molecule with ffMethod = 2 and minimize cartesian "14,to,bb,bs,af".
- set ffMethod to 3 and set terms ""vw,to,el,sf,en" only.
- use show energy or montecarlo

### 2.17.8. gcMethod

method defining how the  $m\_gc$  map is used in the "gc" grid energy calculation. The "gc" method allows to calculate interactions of a molecule with grid energy field representing another molecule ( the first method ), or treat the  $m\_gc$  map as the electron density map.

- 1. "vw" <- default choice: current object interacts with the van der Waals field. Positive values repel, negative attract; Contribution from one non-hydrogen atom is Eatom = 1.\*Egc
- 2. "density" : treats the m\_gc map as positive electron density and pulls the object into it. The contributions of atoms are proportional to atomic number (the number of electrons), hydrogens are ignored: Eatom = -AtomicNumber\*Egc
- 3. "field": uses user-defined atomic field value, which can be set by the set field command and extracted with the Field (as\_) command, as the relative weight of each atom. Anticipates that van der Waals type of the map (attractive negative values, repulsive positive) as in the first method. Eatom = Field(atom)\*Egc

### 2.17.9. highEnergyAction

action taken upon achievement of the maximal allowed number of montecarlo steps resulting in no modification of a stack mnhighEnergy, (it means that conformations are dissimilar to those in the stack and have higher energy). Four actions can be taken:

```
1. "heat"
2. "stackjump" <- default
3. "random"
4. "exit"</pre>
```

### 2.17.10. interruptAction

action taken upon ICM-interrupt (^\ Control backslash).

```
1. = "break loop"
2. = "break all loops" <- default
3. = "exit macro"
4. = "exit to the main macro"
5. = "exit all macros"</pre>
```

### 2.17.11. mfMethod

atom pair selection algorithm used when "mf" energy term is calculated by the show energy, montecarlo, or minimize commands.

Allowed values:

- "intermolecular" (or 1) <- default
- "all" (or 2)

(e.g. mfMethod = 2)

In contrast to the "vw" term, only intermolecular atom pairs are considered by default, since usually intramolecular interactions are calculated with the standard energy terms.

In the "all" mode the atom pairs are taken from the van der Waals interaction lists calculated dynamically in the show energy, montecarlo, or minimize commands. All atom pairs except atoms separated by 1 or 2 bonds (so called 1–2 and 1–3 interactions) and within the vwCutoff distance are taken into account.

```
See also: term "mf", pmf-file, mfWeight.
```

### 2.17.12. minimizeMethod

algorithm used for local energy minimization which takes place in the minimize command, and is a part of one step of a multistep procedure such as montecarlo, ssearch, and convert.

Allowed values:

"conjugate"
 "newton"
 "auto" <- default</li>

"conjugate" means conjugate gradient minimization. Uses analytical first derivatives and takes 6\* *n\_free\_variables* memory.

"newton" – quasi–Newton method. It uses analytical first derivatives and takes *n\_free\_variables\*n\_free\_variables* memory. We recommend this method for energy minimization of small molecules.

"auto" <- default; use the more efficient quasi-Newton if the number of free variables (Nof(v\_//\*) is less than 100 (additional memory requirement of about 2 MB) and switch to the conjugate gradient method if the number of free variables is more than 100.

### 2.17.13. pdbDirStyle

The style of your Protein Data Bank directory/directories. ICM will understand all of the listed styles, including distributions with compressed \*.gz, \*.bz2 and \*.Z files. In all cases, if the s\_pdbDir variable is set correctly, it is sufficient to refer to the file by its four-character code, e.g.

```
read pdb "labc"
```

1. = "labc.brk" <- default
2. = "pdblabc.ent"
3. = "ab/pdblabc.ent"</pre>

Do not forget to set the right pdb-style in your \_startup file.

### 2.17.14. rejectAction

what to do, if the MC procedure rejects mnreject trial conformations in a row. Four actions can be taken:

```
1. "heat" <- default choice
2. "stackjump"
3. "random"
4. "exit"</pre>
```

### 2.17.15. resLabelStyle

style of residue labels invoked by double clicking on the residue or display residue label *rs*\_ command. Possibilities:

1. "A5" <- default choice 2. "Ala 5" 3. "ALA 5" 4. "Ala" 5. "ALA" 6. "Alanine 5"

```
7. "5"
8. "A"
9. " A"
10. "Mol" – displays MOLECULAR name.
```

See also:resLabelShift,atomLabelStyle.

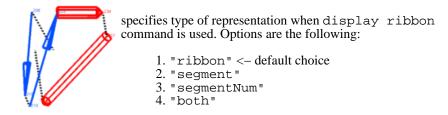
### 2.17.16. ribbonColorStyle

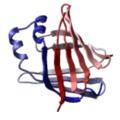
- sets the ribbon coloring scheme.

1 = "none"default. colors by secondary structure type or explicit color2 = "NtoC"colors each chain gradually blue-to-red from N- to C- (or from 5' to 3' for DNA)3 = "alignment"if there is an alignment linked to a protein, color gapped backbone regions gray4 = "reliability"3D gaussian averaging with selectSphereRadius of alignment strength in spaceIf ribbonColorStyle equals to 4, the conserved areas will be colored blue, while the most divergentwill be red, and the intermediate conservation areas will be colored white. Example:

```
nice "leoc.a/"
make sequence a_1.1
read pdb sequence "3pcc.a/"
aa = Align(3pcc_a leoc_a)
ribbonColorStyle=3  # color gaps gray
color ribbon
ribbonColorStyle=4  # see alignment strength
color ribbon
```

# 2.17.17. ribbonStyle





The first choice is a solid ribbon representation. The second, "segment" refers to a style where secondary structure elements are represented by vectors (see Abagyan and Maiorov, 1988).

Note The segment parameters must be **pre-calculated** with the assign sstructure segment command. The segment description is used in fast searches for related topologies in the databank. The last option ("both") will display both representations of the backbone topology.

### 2.17.18. shineStyle

defines how solid surfaces of cpk, skin and grobs reflect light. Possibilities:

1. "white" <- default
2. "color"</pre>

The first option gives a more shiny and greasy look.

### 2.17.19. surfaceMethod

defines how the surface energy is calculated. Options available:

```
1. "constant tension"
2. "atomic solvation" <- default choice
3. "apolar"</pre>
```

**Explanations**:

- 1. "constant tension" means that the energy terms are just the product of the total solvent-accessible surface by the surfaceTension parameter. This term is intended to represent the surface energy if electrostatics takes the solvent polarization energy into account (see electroMethod)
- 2. "atomic solvation" option is designed to evaluate the solvation energy purely on the basis of the atomic accessible surfaces instead of using the proper electrostatic evaluation of the polarization free energy. This fast but approximate scheme was proposed by Wesson and Eisenberg, (1992). Atomic surface parameters derived from the experimental vacuum-water transfer energies are given in the icm.hdt file.
- 3. "apolar" option is designed to evaluate the stabilization energy, which is the difference between denatured and folded states. The "atomic solvation" energy should be used with the van der Waals term while the "apolar" energy takes it into account and should be used without any other energy terms. The "apolar" atomic surface parameters were derived from the experimental octanol-water transfer energies and are given in the icm.hdt file.

Note, that if a part of the system is represented with grid potentials, one needs a special m\_ga grid map for correct calculations of the surface accessibilities.

### 2.17.20. tzMethod

method of imposing and calculating tethers. The three alternatives are the following

- 1. "simple" : equal weight tethers to 3D points
- 2. "weighted" : individual weights are calculated from atomic B-factors by dividing 8\*PI<sup>2</sup> by the B-factor value. All the weights additionally are multiplied by the tzWeight shell variable.
- 3. "z\_only" : tethers are imposed only in the Z-direction towards the target Z-coordinate. These type of tethers pulls a molecule into a z-plane. This may be useful if you are trying to generate a flat projection of a three-dimensional molecule.

Example for the "z\_only" method in which we generate a more or less flat image of a chemical.

### 2.17.21. varLabelStyle

style of labels for free torsions, angles and bonds (i.e. internal variables) display variable label vs command. Possibilities:

1. "greek" <- default choice
2. "name"
3. "value"</pre>

### 2.17.22. visitsAction

what to do, if one stack conformation is overvisited, i.e. mnvisits has been reached. The following actions are allowed:

```
1. "random"
2. "heat" <- default choice
3. "stackjump"
4. "exit"</pre>
```

Explanation of actions:

- "heat" double the simulation temperature
- "stackjump" jump to the conformation of the least visited slot in the stack.
- "random" randomize all free variables according to the mcJump parameter
- "exit" exit the MC procedure

### 2.17.23. vwMethod

specifies the function type of the van der Waals term ("vw"). The following three functions can be chosen:

1. "exact" <- default choice:

 $F_{vw} = A/r^{12} - B/r^6$ . This is the usual van der Waals formula tending to infinity at r close to 0.

2. "soft":

 $F_{soft} = F_{vw}$ , for  $F_{vw} <= 0$ . and  $F_{soft} = F_{vw} * (t/(t+F_{vw}))$  for  $F_{vw} > 0$ . (repulsion).

This form preserves the function for the most populated part of the curve but smoothly reaches the limit t (defined by the vwSoftMaxEnergy real system variable)

3. "old soft":

another smooth approximation with the finite value at r=0, depending on the well depth.

### 2.17.24. webEntrezOption

defines how to interpret the NCBI Entrez links.

- 1. "none"
- 2. "g:GenPept" <- default
- 3. "r:Report"
- 4. "f:FASTA"
- 5. "a:ASN.1"
- 6. "d:Entrez document summary"
- 7. "m:MEDLINE links"
- "p:protein neighbors"
   "n:nucleotide links"
- 10. "t:structure links"
- 11. "c:genome links"

See also: s webEntrezLink, web, show html, write html.

### 2.17.25. wireStyle

style of the display wire mode. The choices are the following:

- 1. "wire" <- default choice
- 2. "chemistry"
- 3. "tree"

Style "chemistry" shows different types of chemical bonds. Style "tree" shows a directed graph of the ICM-molecular tree. Yellow triangle indicates the entry atom of an ICM object. The tree can be rerooted with the write library a\_newEntryAtom command. The topology of the complete tree including the virtual atoms can be shown with the display virtual command.

Note: The "tree" graph does not exist for objects of non-ICM type, e.g. those created by the read pdb command, and this preference will have no effect. The tree representation elucidates the ICM topology graph imposed on molecules and is crucial in the modify command, since it removes a branch up-tree from the specified entry atom, and replaces it by another branch. Use Ctrl-W to toggle between these styles (see set key command). The line width is controlled by the lineWidth parameter.

### 2.17.26. xrMethod

The penalty function of correspondence between observed and calculated structure factors.

- 1. "corr Fc:Fo" <- default
- 2. "corr Fc2:Fo2"

# 2.18. Tables (structures)

The following predefined icm-shell tables are collections of different icm-shell-objects related to a certain topic. Note that these tables (as opposed to user-defined ICM tables) usually only have the header section. You can show and list them. You can also change any table element by the usual icm assignment:

Examples:

```
IMAGE.color = yes  # this member is a logical
IMAGE.stereoBase = 2.5  # redefine real distance between stereo panels
```

### 2.18.1. FILTER

contains filters which can be applied to the input stream in the read command. Components have names corresponding to standard file name extensions; their string value is a unix filter. Token **%s** is a placeholder for the file name. The provided defaults can be redefined in your \_startup file. You can also add your own extensions and filters by doing the following:

z = "pcat %s" # define the action for the unix packed files group table append FILTER header z # append new filter to the structure

The mechanism ICM employs allows to keep the transformed files intact and avoid creating temporary files when possible (e.g. uuencode unix command always creates an output file). Existing extensions and defaults are given below. You may need to redefine the defaults by adding the exact path to the utility or using alternatives.

#### FILTER.Z

allows you to read the compressed files (\*.Z) directly leaving the compressed file intact. The default value: "zcat %s". If you do not have zcat utility, try

FILTER.Z = "uncompress - c %s"

#### FILTER.gz

The default value is "gunzip -c %s".

#### FILTER.uue

The default value is

```
"sed 's:begin .*:begin 600 /tmp/UUPtm:' %s | uudecode &cat /tmp/UUPtm &rm -f /tmp/UUPtm"
```

This works for UNIX file system, write your own on the PC, if needed.

### 2.18.2. FTP

table which controls reading from ftp.

#### FTP.createFile

(default no ). This flag is not active yet. The file is always created in the s\_tempDir directory.

#### FTP.keepFile

(default no ). If yes , the temporary file is kept in the  $s\_tempDir$  directory. Otherwise the file is deleted.

#### FTP.proxy

string path to the proxy server for connections through firewall. Default: "" (empty string).

### 2.18.3. GRAPHICS

display parameters for different graphics representations.

#### **GRAPHICS.ballRadius**

radius (in Angstroms) of a small ball displayed as a part of ball or xstick graphical representations of a molecule.

Default (0.15)

#### GRAPHICS.dnaBallRadius

DNA bases in ribbon representation are shown as balls controlled by the above real parameter. You can undisplay them with the: undisplay ribbon base command.

Default: 1.5

#### GRAPHICS.dnaRibbonRatio

real ratio of depth to width for the DNA ribbon.

Default: 0.3

#### GRAPHICS.dnaRibbonWidth

real width (in Angstroms) of the DNA ribbon.

Default: 2.

#### GRAPHICS.dnaRibbonWorm

logical which, if yes, makes the DNA backbone ribbon round, rather than rectangular.

Default: no

#### GRAPHICS.dnaStickRadius

real radius of the sticks representing bases in DNA ribbon.

Default: 0.72

#### **GRAPHICS.quality**

integer parameter controlling quality (density of graphical elements) of such representations as cpk, ball, stick, ribbon. Do not make it larger than about 20 or smaller than 1. This parameter supersedes the previous ballQuality parameter.

We recomment to make this parameter at least 15 if you want to make a high quality image. You can also increase the number of image resolution by making the image window 2,3,4 times larger (in the example below it is 2 times larger) than the displayed window.

```
GRAPHICS.quality = 15
display ribbon
# press Ctrl-D for the fog effect, move clipping planes, change fogStart
write image png window=2*View(window)
```

Default: 5.

#### GRAPHICS.rainbowBarStyle

determines if and where the color bar will appear after a molecule is colored by an array. Coloring by an array is one of the options of the display and color commands.

1. = "text at left" <- default choice

- 2. = "text at right"
- 3. = "no text"
- 4. = "no bar"

The bar can be dragged (use middlebutton), removed (point into the bar and press BACKSPACE), just like a string label. To assign your own numbers to the bar, you may choose option "no text" and use several display s\_label commands. The bar, if displayed, is exported to TIF, RGB images and postscript.

#### GRAPHICS.resLabelDrag

if yes, enables dragging of the displayed residue labels with the middle mouse button. The labels can be reset to their initial positions with the set residue label distance rs\_

command. The initial position is defined by the relative displacements of  $\{0. 0. 0.\}$  from the special "residue label-carrying" atom of the residue, see the set label as\_command. See also resLabelStyle

Default (no).

#### **GRAPHICS.ribbonRatio**

real ratio of depth to width for the protein ribbon.

Default: 0.3

#### GRAPHICS.ribbonWidth

real width of the protein ribbon.

Default: 1.

#### GRAPHICS.ribbonWorm

logical parameter, if yes, makes the ribbon round, rather than rectangular.

Default: no

#### **GRAPHICS**.selectionLevel

preference for the selection level of as\_graph selection. The atoms, residues, molecules or objects selected interactively in the graphics window are automatically stored in the as\_graph variable. The preference may have the following values.

```
GRAPHICS.selectionLevel = "atom"
    1 = "object"
    2 = "molecule"
    3 = "residue"
    4 = "atom" # default
    5 = "variable"
```

The GRAPHICS.selectionLevel can be switched either interactively, e.g.

```
GRAPHICS.selectionLevel = 3
```

or from GUI by selecting the level combo box with the following choices: O (object), M (molecule), R (residue), x (atom), or an icon of a torsion (variable).

#### **GRAPHICS.selectionStyle**

preference for the style in which the graphical selection is shown. The preference may have the following values.

```
GRAPHICS.selectionStyle = "color"
    1 = "none"
    2 = "cross"
    3 = "color" # the default choice
    4 = "cross and color"
```

In the 1-st mode ( "none") only a single selection mark is shown. It is convenient when you do not want multiple selection marks to overwhelm the image. The 3-rd mode is incovenient if you want to try different

colored displays for the selected fragments.

#### **GRAPHICS**.stickRadius

radius (in Angstroms) of a cylinder displayed as a part of stick or xstick graphical representation of a molecule.

Default (0.4).

#### **GRAPHICS.wormRadius**

radius of coiled segments (i.e. those where the secondary structure is marked as "\_") of a polypeptide chain in ribbon representation.

Default (0.3).

### 2.18.4. GRID

parameters for the grid energy calculations (see also "gh,gc,ge,gs,sf" energy terms).

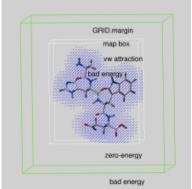
#### GRID.gcghExteriorPenalty

A preference to allow automatically impose a repulsive penalty outside the area covered by the van der Waals maps (m\_gc and m\_gh).

1. = "repulsive" <- the default
2. = "zero"</pre>

In the default mode a volume penalty is imposed automatically outside the map box expanded by the GRID.margin. The penalty potential is set to the GRID.maxVw value.

#### **GRID.margin**



real parameter determining the extra penalty-free space around the map bounding box if GRID.gcghExteriorPenalty = "repulsive" (see above). For any atom which gets outside the map-bounding box expanded by GRID.margin, its grid van der Waals energy ( "gc" or "gh" ) is penalized by the GRID.maxVw value. This is the same penalty value which atoms get if they severely clash with other atoms.

Therefore, if you set up grid energy calculations it is essential either to create a big enough box or set a sufficient margin to allow ligand rearrangements near the receptor surface. If GRID.margin is very large,

your ligand will be "on the loose" and may spend too much time flying in open air. It is recommended that the margin is not larger than the diameter of your ligand.

Default: 0.00 A

#### **GRID.maxEl**

real truncation parameter. Default: 20.0 kcal/mole.

#### **GRID.minEl**

Default: -20.0 kcal/mole.

#### GRID.maxVw

The truncation level of the van der Waals repulsion energy precalculated in the "gc" grid energy term. This number also is used as a penalty for the atoms outside the map box expanded by GRID.margin.

Default: 3.0 kcal/mole.

### 2.18.5. GROB

Parameters related to graphics objects. See also the Grob family of functions.

#### GROB.relArrowSize

a real relative arrow size ([0.,1.]). Default: 0.2.

#### **GROB.arrowRadius**

a real arrow radius in Angstoms used by the Grob( "ARROW", R\_) function. Default: 0.5.

#### **GROB.relArrowHead**

a real ratio of the arrow head radius to the arrow radius. This parameter is used by the Grob( "ARROW",  $R_{}$ ) function. Default: 3.0.

### 2.18.6. GUI

table contains some settings for the GUI. Most of the settings are stored automatically in the s\_userDir + "/config/icm.cfg" file

#### GUI.workspaceTabStyle

allows to change the style of ICM-object tabs created in the workspace panel of ICM GUI.

1. = "icon title" # default 2. = "icon" 3. = "title"

### 2.18.7. IMAGE

table contains settings used by the following commands creating image files:

- •write image,
- write postscript,
- display movie image.

#### **IMAGE.quality**

this integer parameter allows to improve quality of vectorized postscript images saved by the write postscript command. Actually this parameter only changes one number in the header of a postscript file. You can also manually edit the file to correct this number. This number defines the number of divisions of larger triangles into smaller ones accompanied by interpolation of colors which occurs during printer interpretation of the postscript stream to provide smooth continuous transitions. The optimal value of this parameter depends on the maximal triangle size. It may grow as large as 100 for a single triangle on a page. Typically for a molecular image with molecular surface IMAGE.quality=3 is sufficient.

Important. Do not set the parameter to values higher than 5 for the molecular image, your printer will die!

Default: 3

#### **IMAGE.printerDPI**

this integer parameter the printer resolution in Dot Per Inch (DPI). Important for the write image postscript command.

Default: 300

#### IMAGE.lineWidth

this real parameter specifies the default line width for the postscript lines.

Default: 1.0

#### **IMAGE.scale**

real variable. If non zero, controls the image scale with respect to the screen image size. The screen image resolution (or Dots-Per-Inch) is usually 72. Let's assume printer DPI to be 300 (see the IMAGE.printerDPI parameter). In this case IMAGE.scale=1. will make the printed image the same pixel size (which is about 4 times smaller) than the screen image. For pixel images saved by write image postscript command integer IMAGE.scale values (2., 3., 4.) are preferable. That is what auto mode (IMAGE.scale=0.0) is trying to do. This consideration is NOT important for the vectorized postscript images created by the write postscript command.

Default: 0.0 (i.e. auto mode: maximum size fitting the page in given IMAGE.orientation)

#### **IMAGE.stereoBase**

real variable to define the stereo base (separation between two stereo panels) in the write image postscript and write postscript command.

Default: 2.35 inches, (~ 60mm)

#### IMAGE.stereoAngle

real variable to define stereo angle (relative rotation of two stereo images) in the write image postscript and write postscript command.

Default: 6.0 degrees.

#### IMAGE.gammaCorrection

real variable to to lighten or darken the image by changing the *gamma* parameter. A gamma value that is greater than 1.0 will lighten the printed picture, while a gamma value that is less that 1.0 will darken it. You may adjust your gamma correction parameter for your printer with respect to your display and add this setting to the \_startup

Default: 2.0

#### IMAGE.color

logical to save color or black\_and\_white ('bw') images. You can override this parameter by using the explicit bw option in the write image command.

Default: yes

#### **IMAGE.compress**

logical to toggle simple lossless compression, standard for .tif files. This compression is required to be implemented in all TIFF-reading programs.

Default: yes

#### IMAGE.generateAlpha

logical to toggle generation of the alpha (opacity) channel for the SGI rgb, tif and png image files to make the pixels of the background color transparent.

Be careful. The alpha channel is set to 1. for every pixel in your image which has the same color as the background. Therefore there is a danger that the same color will be accidently used inside your image. If you nevertheless want to generate the alpha-channel, use a rare color your background (not black, but rather green, e.g. rgb =  $\{0., 0.976, 0.\}$ .

Default: yes

#### IMAGE.stereoText

logical to make text labels for only one panel or both panels of the stereo diagram.

Default: yes

#### **IMAGE.previewer**

a string parameter to specify the external filter which creates a rough binary (pixmap) postscript preview and adds it to the header of the ICM-generated high resolution bitmap or vectorized postscript files saved by the write image postscript, and write postscript, respectively. This preview information is compliant with EPSI (encapsulated Postscript interchange file format) and is useful to see a draft image instead of a empty rectangle upon inclusion of the postscript file into other drawing and imaging software like IRIS showcase.

Default: "gs -sDEVICE=pgmraw -q -dNOPAUSE -sOutputFile=- -r%d -- %s"

#### IMAGE.previewResolution

integer resolution of the rough bitmap preview added to the vectorized postscript file in *lines per inch*. Recommendations:

- 10 very rough ( 1/10 th of an inch )
- 20 a reasonable preview but no fine details
- 30 a fine preview, do not increase it any higher since the file will become too large.

#### **IMAGE.orientation**

preference to specify image orientation.

- 1. = "portrait" <- default
- 2. = "landscape"
- 3. = "auto"

Default: "portrait"

#### **IMAGE.paperSize**

preference to specify paper size.

1. = "Letter (8.5x11")" <- default 2. = "Legal (8.5x14")" 3. = "11x17"" 4. = "A4 (210x297mm)" 5. = "A3 (297x420mm)"

Default: "Letter (8.5x11")"

#### IMAGE.rgb2bw

rarray of 6 elements defining translation of rgb colors into black and white ('bw') grades. The array is {RED\_scale, GREEN\_scale, BLUE\_scale, RED\_bias, GREEN\_bias, BLUE\_bias} and the default values are {0.3125, 0.5, 0.1875, 0., 0., 0.}.

### 2.18.8. LIBRARY

table containing string paths of the icm parameter files, which are loaded by the read library [mmff] command. The library files will be taken from the s\_icmhome directory if no explicit directory is provided. Extensions are automatically added. Defaults:

```
LIBRARY.bbt="icm" # bond bending types
LIBRARY.bci="icm" # mmff bond charge increments
LIBRARY.bci="icm" # bond stretching types
LIBRARY.cnre"icm" # colors, gui controls
LIBRARY.cmp="icm" # amino-acid comparison matrix
LIBRARY.cnt="icm" # distant restraint types
LIBRARY.cod="icm" # atom codes
LIBRARY.hbt="icm" # hydragen bonding types
LIBRARY.hbt="icm" # hydration types
LIBRARY.hbt="icm" # hydration types
LIBRARY.lps="icm.lps" # loop database, rebuilt with write model [append]
LIBRARY.mmbt= "mmff" # mmff bond bending
LIBRARY.mmbt= "mmff" # mmff bond stretching
LIBRARY.mmtot= "mmff" # mmff torsions
LIBRARY.mmvwt= "mmff" # mmff van der Waals
LIBRARY.tor="icm" # variable restraint types
LIBRARY.tor="icm" # torsion types
LIBRARY.tor="icm" # torsion types
LIBRARY.tor="icm" # torsion types
LIBRARY.res={"icm" # van der Waals types
LIBRARY.res={"icm", "usr"}
```

Example:

```
LIBRARY.res=LIBRARY.res // "./benz.res" # just append
LIBRARY.cod="./newCodes.cod"
read library
```

### 2.18.9. OBJECT

Controls atom requisites which are written to a file in the write object command. Extensions are automatically added. Defaults:

OBJECT.bfactor =yes OBJECT.charge =yes OBJECT.occupancy=yes OBJECT.site =yes OBJECT.display =no OBJECT.library =no OBJECT.auto =no

Example:

```
OBJECT.auto = no

OBJECT.display = yes

read object "crn"

display ribbon a_/1:40

set plane 2

display cpk a_/12

write object "tm" # graphics and planes are written

delete a_*.

read object "tm"
```

### 2.18.10. PLOT

Contains settings used by the plot command. All real sizes are expressed in the Postscript "points" equal to 1/72" ( about 1/3 mm ).

#### PLOT.numberOffset

integer offset for the X-coordinate with the number option. This option is used in a number of macros generating multisection plots for amino-acid sequences.

Default (0).

#### PLOT.fontSize

real font size. Any reasonable number from 3. (1 mm, use a magnifying glass then) to 96.

Default (10.0).

#### PLOT.lineWidth

real line width for graphs (not the frame and tics)

Default (1.0).

#### PLOT.markSize

real mark size in points. Allowed mark types: line, cross, square, triangle, diamond, circle, star, dstar, bar, dot, SQUARE, TRIANGLE, DIAMOND, CIRCLE, STAR, DSTAR, BAR. Uppercase words indicate filled marks.

Default (1.0).

#### PLOT.Yratio

real aspect ratio of the ICM plot frame. Using link option of the plot command is equivalent to setting this variable to 1.0. If PLOT.Yratio is set to 0., the ratio will be set automatically to fill out the available box optimally.

Default (0.8).

#### PLOT.logo

logical switch for the ICM-logo on the plot.

Default (yes).

#### PLOT.color

logical to generate a color plot. Usually it does not make sense to switch it off because your b/w printer will interpret the color postscript just fine anyway.

#### **PLOT.orientation**

preference for the plot orientation. Currently inactive. Default (yes).

#### PLOT.seriesLabels

preference to indicate position of a series/color legend inside the plot frame. You can provide individual names for each series in the optional string array argument of the plot command. (e.g. plot M\_XY1Y2 {"Title","X","Y","Ser 1","Ser 2"}) Available choices:

- 1 = "none"
- 2 = "right" <- default choice
- 3 = "left"
- 4 = "top"
- 5 = "bottom"

#### **PLOT.font**

preference for the title/legend font. The font size can only be redefined by editing the \*.eps file (search for the number before the scalefont string). Available choices:

- 1 = "Times–Bold"
- 2 = "Times–Roman" <– default choice
- 3 = "Helvetica"
- 4 = "Courier"
- 5 = "Symbol"

#### **PLOT.labelFont**

preference for the data point label font. You can also redefine the font size with the PLOT.fontSize variable. Available choices:

- 1 = "Times-Bold"
- 2 = "Times-Roman" <- default choice
- 3 = "Helvetica"
- 4 = "Courier"
- 5 = "Symbol"

#### PLOT.rainbowStyle

preference defining the color spectrum used by the plot area command. This command lets you plot a function of 2 arguments and show the function value by color. By default the plot command uses the minimal and maximal values of the provided matrix. You can enforce the range with the color option. Available choices:

- 1 = "black/white"
- 2 = "blue/white/red" <- default choice
- 3 = "blue/rainbow/red"

Example:

```
read matrix # def.mat is the default one
PLOT.rainbowStyle=1
plot area def display # grey-scale, automatical min and max
PLOT.rainbowStyle=3
plot area def color={-10.,0.} display # enforce new range
PLOT.rainbowStyle=2
plot area def transparent={-2.,8.} display
# low values - blue, middle [-2.,8.] - invisible, large red
```

#### PLOT.box

rarray of the origin and relative sizes of the ICM plot frame: { X\_origin, Y\_origin, X\_size, Y\_size }. Box {0. 0. 1. 1.} fits the page optimally.

Default ({0. 0. 1. 1.}).

### 2.18.11. SITE

This table contains parameters and preferences used to display the sites, or important residues.

#### SITE.defSelect

string of significant site types (shown as one letter abbreviations) Sequence identity in the alignment positions which have one of those sites is additionally rewarded in the alignment score calculation.

Default: "ABFGLMstepm"

#### SITE.labelOffset

(default 5. A) the real offset of the site label with respect to the residue label atom.

#### SITE.labelStyle

the style preference of the displayed site information:

- 1. "none"
- 2. "symbol" # one letter symbol, see site.
- 3. "TYPE" # < --- default choice

4. "RES.TYPE"

- 5. "comment"
- 6. "full"

#### SITE.labelWrap

0.5 (inactive)

#### SITE.showSeqSkip

the string of the site types skipped in the show sequence (or alignment) commands.

### 2.18.12. WEBLINK

This table contains definitions of types of web links used in the web, show html, and write html commands. The table is read from "WEBLINK.tab" file from the \$ICMHOME directory. Change this file for your own definitions. The weblink specification is used to extend the argument string substituted for %s (e.g. "IL2\_HUMAN" element of the table array linked according to the type

SP %s "http://www.aaa?%s" will be transformed into the <a href="http://www.aaa?IL2\_HUMAN">IL2\_HUMAN</a> link. If %Ns specification is used, only N characters of the argument string will be retain in the link. For example,

PDB %s "http://www.pdb?%4s" and lxyz\_a15\_25 (specifying chain and residue range) will be translated into

<a href="http://www.pdb?lxyz">lxyz</a>\_a15\_25 which in your browser will look like
this:

"AUTO" is another type which can be used in the link S\_ "TYPE" ... expression. In this case the DB type is automatically recognized according the database reference string pattern (see also WEBAUTOLINK). An example table:

#>T WEBLINK #>-DB-----DR--LINK------PDB %s http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=s&form=6&uid=pdb|%4s|&Dop NCBI g%s http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=s&form=6&uid=%s&Dopt=g EMBL %s http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=s&form=6&uid=emb|%s|&Dopt SP %s http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=s&form=6&uid=emb|%s|&Dopt= SPA %s http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=s&form=6&uid=sp|%s|&Dopt= PROSITE %s http://saturn.med.nyu.edu/srs/srsc?[PROSITE-acc:%s] MED %s http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=m&form=6&uid=%s

#### Example:

read table "seqcomp.tab" #contains references to different databases
web SR link SR.NA1 "PDB" SR.NA2 "AUTO"

### 2.18.13. WEBAUTOLINK

This table contains definitions of web link string patterns for automatic recognition in the web, show html, and write html commands. The table is read from the "WEBLINK.tab" file in the \$ICMHOME directory. Change the file for your own definitions. Recognition is not perfect because the patterns overlap.

Example:

read table "seqcomp.tab" #contains references to different databases web SR link SR.NA1 "AUTO" SR.NA2 "AUTO"

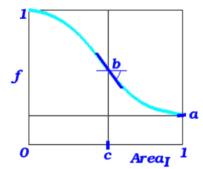
# 2.19. Other shell variables

### 2.19.1. defCell

the real array of the default cell parameters. This definition is used in the Resolution and MaxHKL functions if cell parameters are not provided as arguments.

Default: {1. 1. 1. 90. 90. 90.}

### 2.19.2. accFunction



the real array of the solvent accessibility penalty parameters (as described in Batalov and Abagyan, 1999).

It contains the values of *a*, *b*, *c* and *E* damping parameters for aminoacid substitution scores. Generally, if a residue is completely buried (*Area=0*), its substitution scores will be used without changes. If it is completely exposed, its substitution scores will be multiplied by the minimal possible value of *a*. Between these cases the substitution scores are modulated by a smooth ("arctangent") function with a saddle point at *Area=c*, where the slope will be -b. The fourth parameter is reserved for development.

This definition is effectively implemented in the  $Align(seq_1 seq_2 area)$ }, Score functions and find database command.

Default: {0.33 2.35 0.211, -15.0}.

See also: alignMethod.

### 2.19.3. gapFunction

the real array of the gap penalty parameters, which represent a piecewise-linear concave function (as described in Batalov and Abagyan, 1999).

**ATTENTION**: at the present time this gapFunction is only active when alignMethod =2.

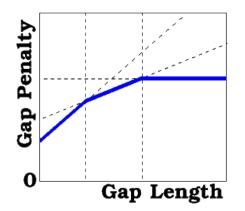
The first two values replace gapOpen and gapExtension traditional values. If present, the third element of the array represents the length of the gap, starting at which further gapExtensions become equal to the fourth element of the array. Likewise, if more elements are present, they represent pairs of the threshold lengths of the gap and the new **gapExtensions** values. For example,

gapFunction = {2.4 0.15 10. 0.05 20. 0.}

means that

• gap penalty=2.25+0.15\*L for L={ 0..10} (and for L=1 it is 2.4= gapOpen ),

• gap penalty=4.25 for L>20



The calculations are fastest for the traditional two-element gapFunction. The three- or four-element gapFunction invokes the optimized routines and is 50-70% slower. The general kind gapFunction costs approximately 70-90% additional time for every pair of gapFunction values. If the last gapExtension is zero, it may be omitted. This definition is effectively implemented in the Align, Score functions and find database search command.

Default: {2.4 0.15}.

**Recommended** (put it in your \_startup file): gapFunction = {2.4 0.15 10.}

This set will produce fast and structure-like alignments.

See also alignMethod, and accFunction (the accessibility attenuation parameters).

### 2.19.4. I\_out

an integer array in which the output of some commands is stored.

### 2.19.5. M\_out

matrix in which the output of some commands is stored.

### 2.19.6. R\_out

real array in which the output of some commands is stored.

### 2.19.7. S\_out

string array in which the output of some commands is stored.

### 2.19.8. swissFields

string array of SWISS-PROT fields to be read by default in read sequence swiss

### 2.19.9. readMolNames

string array in which the SDF-file comment fields containing database compound identifier and description are preset. There is a standard place where database compound identifier should be stored in SDF (MOL)-files. This is the first line of the entry. However most of the database providers got used to leaving this line empty. Instead they put identifier and description in the end of the file in the following fashion:

```
M END

> <CAT_NO>

R150002

> <NAME>

(5-OXO-HEXAHYDRO-PYRANO[3,2-B]PYRROL-1-YL)-ACETIC ACID METHYL ESTER

$$$$
```

In this particular case before using such database set

```
readMolNames = {"<CAT_NO>" "<NAME>"} # useful for Sigma-Aldrich files
```

Another example:

```
readMolNames = {"<CODE>" "<IUPAC_NAME>"} # useful for ACD database
```

### 2.19.10. Named Atom/Residue/Molecule/Object Selections

Examples:

cc = a_//ca		# C:	reat	ced	name	d selection	variable	CC
show cc a_/3:15	#	use	it	in	the	expression		

In this case the named selection **cc** is a true ICM–shell variable, not just an alias for the Ca selection. Please do not confuse it with another useful mechanism which allows you to use a **string** in a selection. This mechanism is used in scripts and macros.

Example:

cc = "a\_//ca"  $\ \ \, \#$  in this case cc is a string, not a selection show \$cc a\_/3:15 # \$cc is replaced by a\_//ca before parsing

### 2.19.11. as\_out

an atom/residue/molecule/object selection variable where some commands or functions store their output:

- Rmsd
- Srmsd
- superimpose
- set tether

```
• show drestraint
```

• show tethers

If atoms a\_1./3/ca,c,n relate to atoms a\_2./45/ca,c,n, then the first set will end up in as\_out and the second in as2\_out .

### 2.19.12. as2\_out

the second set of atoms ( selection ) returned by the following commands and functions:

- Rmsd
- Srmsd
- superimpose
- set tether
- show drestraint
- show tethers

See also as\_out above.

# 2.19.13. Named Selections of Internal Variables (Dihedrals, Angles and Bonds)

### 2.19.14. vs\_out

The variable selection where some commands or functions store their output:

• read variable saves a selection of loaded variables;

# 2.20. Commands

### 2.20.1. alias

alias abbreviation word1 word2 ...

create alias

alias delete abbreviation

delete alias

It is important that the abbreviation is not used in the ICM-shell. The same names can not be given later to ICM-shell objects.

Alias may contain arguments \$0, \$1, \$2, etc. ICM–shell will pick space–separated words following the alias name and substitute \$1, \$2, etc. arguments by the specified argument. \$0 stands for all the arguments after the alias name.

Examples:

alias seq sequence alias delete seq alias dsb display a_//ca,c,n	<pre># seq will invoke sequence # delete alias name seq # abbreviate several words to # reduce typing efforts # aliases with arguments</pre>
alias NORM (\$1-Mean(\$1))/Rmsd(\$1) show NORM {6,7,8,4,6,5,6,7,5,6}	# make sure there is no space

### 2.20.2. align

#### align number: renumber residues sequentially

align number rs\_residuesToBeRenumbered [ i\_firstNumber ]

align number ms\_chainsToBeRenumbered [i\_firstNumber]

renumber selected residues, or residues in selected molecules or objects sequentially in all of them from starting one or the specified first number. May be useful to deal with messy numbering in some pdb–files. Example:

align number ms\_chainsToBeRenumbered seq\_master [ i\_offset ]

renumber the residues of the selected molecule according to *seq\_master* master sequence which is aligned to the sequence of the selected chain. The alignment (pairwise or multiple) need to be linked to the molecule/chain and both the chain sequence and the master sequence need to be covered by the alignment. The molecular sequence can be generated with the make sequence [*ms\_chainsToBeRenumbered*] command. The can be either aligned anew the the master sequence with the Align function or appended to a multiple sequence alignment using alignment projection.

This command may be useful in cases in which a structural model does not represent the entire sequence because of omitted loops, N- and C- termini, while you still want to keep the numbering according to the full master sequence. You might want to use the command also on models by homology generated with the build model command.

Example:

```
seqmaster = Sequence("ACDEFGHIKLMNPQRST")
buildpep "--DEFGH-----PQRST" # dashes are skipped
make sequence a_1 name="seqmodel" # sequence is auto-linked
a = Align(seqmodel,seqmaster) # linked alignment
align number a_1 seqmaster
# Info> residues of a_def.m renumbered by sequence 'seqmaster' from alignment 'a'
display residue label
```

#### align: ICM multiple alignment algorithm

```
align ali_SequenceGroupName [ tree= s_epsFileName ]
```

make a multiple alignment of specified sequences a sequence group resulting from the group sequence *s\_groupName* command. For pairwise alignment use the Align(*seq1* seq2) function. The algorithm includes the following steps (inspired by corridor discussions with Des Higgins, Toby Gibson and Julie Thompson):

- 1. align **all** sequence pairs with the ICM ZEGA algorithm, and calculate pairwise distances between each pair of aligned sequence with the Dayhoff formula, e.g. the distance between two identical sequences will be 0., while the distance between two 30% different sequences will be around 0.5. The distance goes to an arbitrary number of 10. for completely unrelated sequences. The distance matrix  $D_{ij}$  can later be extracted from the alignment with the Distance( $ali_$ ) function.
- 2. build an evolutionary tree from  $D_{ij}$  with the "neighbor-joining algorithm" of Saitou, N., Nei, M. (1987) to determine the order of the alignment and calculate relative weights of sequences and profiles from the branch lengths. The tree will be saved in the file defined by the tree option (aligTree.eps file by default). The so-called Newick tree description string will be saved in  $s_{out}$ .
- 3. traverse the tree from top to bottom, aligning the closest sequences, sequence and profile or two profiles. After each Needleman and Wunsch alignment, build the profile.
- 4. generate the final neighbor-joining evolutionary tree and write the PostScript file with the tree to disk.

#### Examples:

#### EST, DNA alignment and assembly

align new ali\_sequenceGroup [ seq\_seed ]

multiple alignment of ESTs and genomic DNA and consensus derivation. This command uses the external the sim4 program to generate pairwise alignments between expressed DNA sequence and a genomic sequence. The sim4 program can be downloaded from the http://globin.cse.psu.edu/globin/html/docs/sim4.html site.

The procedure has the following steps:

- sequences are sorted by length
- the longest sequence is chosen as the seed sequence unless it is explicitly provided
- the longest sequence from the remaining set is aligned to the seed sequence using the external sim4 program.
- the output of this program is parsed and translated into the icm alignment

# Consensus • the consensus sequence is created and 26006s becomes the master sequence TCATCACCAACGACCTCCTGGGGAAGCGCCC TC<mark>ATCACCAACG</mark>ACCTCCTGGG.AAGCGCCC 56862s • the procedure is repeated until all the 4908s sequences are processed u cons TCATCACCAACGACCTCCTGGG.AAGCGCCC • the multiple sequence alignment is further cleaned to compress spurious # Consensus gaps when possible. This cleaning 94939s C makes the consensus much more 26006s CAGA.CTCCCT.CCT.CANT..GGAGATTT CAGAATTCCCTTCCTNAAATTGGGANTTTT compact. 56862s CAGA.TTCCCT.CCT.CAAT.GGAGATTT 4908s The result of this command is best displayed with AGA.YTCCCT.CCT.MAAT..GG<mark>A</mark>GWTTT u cons the show color ali\_command. # Consensus TTCCAGTGGTGCCTAAGAACGTGGCCTTTGT TTCCAGTGGTGCCTAAGAACGTGGCCTTTTT 94939s C 26006s TTCCAGTGGTGCCTAAGAACGTGGCCTTTGT 4908s TTCCAGTGGTGCCTAAGAACGTGGCCTTTKT

u cons

### An example:

```
read sequence "http://www.ncbi.nlm.nih.gov/UniGene/" + \
    "download.cgi?ID=5198/tt> #
read sequence "../Hs5198"
group sequence unique u # squeeze out obvious redundancies and form group 'u'
align new u
                        # form multiple alignment and build consensus
show color u
```

See also:

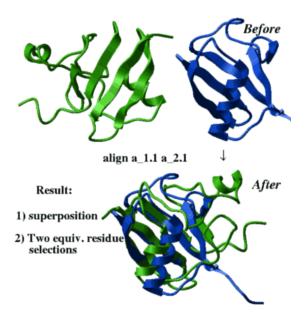
- filtering, group sequence unique="..."
- Trans ()
- show [color] *ali*

### align two molecules by their backbone topology

align [distance] ms\_1 ms\_2 [i\_windowSize=15] [r\_seqWeight=0.5]

This command finds the residue *alignment* (or residue-to-residue correspondence) for two arbitrary molecules having superimposable parts of the backbone conformations. The structural alignment identification and optimal superposition is primarily based on the C-alpha-atom coordinates, but the sequence information can be added with a certain weight (the default value of r seqWeight is 0.5 which was found optimal on a benchmark). The structural alignment algorithm is based on the ZEGA (zero-end-gap-alignment) dynamic programming

procedure in which substitution scores for each i,j-pair of residues contain two terms:



- structural similarity in a *i\_windowSize* window between two fragments surrounding residues i and j, respectively. This similarity is calculated as local Rmsd of the residue label atoms (these atoms are C-alpha atoms by default but can be reset to other atoms with the set label command, e.g. set label a\_\*.//cb). If the option distance is specified the deviation of the interatomic distances between equivalent *pairs* of atoms (so called *distance rmsd*) is calculated instead of a more traditional root-mean square deviation between atom coordinates of equivalent atoms. The latter method is less accurate but an order of magnitude faster.
- sequence similarity (if  $r\_seqWeight > 0$ .). Average local sequence alignment score in the *i\\_windowSize* window is calculated for i,j-centered pair of fragments. In this sense this sequence similarity is different from the one used in pure sequence alignment (see the Align function), in which just the i,j residue pair is evaluated. The default value of  $r\_seqWeight$  of 0.5 is rather mild (about a half of the structural signal).

### The output:

- ali\_out contains structural alignment (if sequences linked to the molecules do not exist, they will be created on the fly). The alignment can be further edited with the interactive alignment alignment editor.
- as\_out contains the residue selection of the aligned residues in the first molecule
- as2\_out contains the residue selection of the aligned residues in the second molecule
- M\_out , the matrix of local structural/sequence similarity in a window is retained and can be visualized by:

```
make grob color 10.*M_out name="g_mat # x,y,z scales
display g_mat
# or
plot area M_out display grid link
```

See also:

- Align( *seq\_1 seq\_2* distance | superimpose ). This function creates the first unrefined structural alignment as described above.
- find alignment which refines initial structural alignment.

The overall result of the align command is equivalent to:

```
a = Align(... superimpose ) # superposition/RMSD based local str. alignment
a = Align(... distance ) # distance RMSD based local str. alignment
find a superimpose 4.0 0.5
```

### Example:

```
read object s_xpdbDir + "/lbrl.b/"
read object s_xpdbDir+"/lnfp"
rm a_*.!A
display a_*.//ca,c,n
color molecule a_*.
align a_2.1 a_1.1
center
show String(as_out) String(as2_out)
color red as_out
color blue as2_out
show ali_out
```

### align heavy command for multiple alternative structural alignments.

### align heavy rs\_1 rs\_2 [r\_rmsd] [i\_windowSize [i\_minFragment]] [r\_elongationWeight]

This method, as opposed to the default align ms\_1 ms\_2 generates many possible solutions and does not depend on sequential order of the secondary structure elements. However, it leads to a combinatorial explosion and is intrinsically less stable computationally, and generally requires more time. The command finds the optimal 3D superposition between two arbitrary molecules/fragments (two residue selections  $rs_1$  and  $rs_2$ ).

The procedure generates structural fragments of certain initial length and superimposes all of them to calculate the structural similarity distance. Then the "islands" of similarity are merged into larger pieces. This process is controlled by the following arguments: *i\_windowSize* is the residue length of structural fragments for the initial fragment superposition. Fragment pairs with the rms deviation less than *r\_rmsd* are then combined, giving composite solutions of total residue length larger than *i\_minFragment*. Acception or rejection of the composite solutions is governed by the following score (the smaller, the better)

### score = rmsd - (1.37 + Sqrt(1.16 \* length - 15.1)), length >= 14

If *length>14*, we use linear extrapolation of the score dependence:

score = rmsd - (1.37 + 1.068\*(length-13))

The score is required to be less than  $r_rmsd$ . Practically, for longer fragments one can find much larger RMS deviations according to the length correction of the score.

Defaults:

- *r\_rmsd* = 1. A
- $i_windowSize = 15$  residues
- *i\_minFragment* = *i\_windowSize*
- r\_elongationWeight=0.1

There may be several different reasonable solutions. All the solutions are sorted, shown and stored in the memory. The two output selections as\_out and as2\_out contain the best scoring solution. Any solution can be loaded and displayed. Additionally, a residue alignment is created for each solution. The decision about which residues are aligned is based on the overall score described above for the of combined fragments.

See also: How to optimally superimpose without the residue alignment

Example:

```
read pdb "4fxc"
read pdb "lubq"
display a_*.//ca,c,n
color molecule a_*.
align heavy a_1.1 a_2.1 12 1.5 .1
center
                           # load the second best solution
load solution 2
color red as_out
color blue as2_out
for i=1,10
  load solution i
  color molecule a *.
  color red as_out
  color blue as2 out
                            # rotate and hit 'return'
  pause
```

endfor

**Note**. Increase *i\_minFragment* parameter (12 in the above example) to something like 20 if the program hangs for too long. Interrupt execution with the ICM–interrupt (**Ctrl** ) if you want only the top solutions.

### 2.20.3. assign

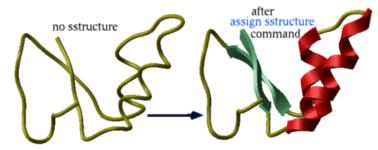
### assign sstructure: derive secondary structure from a pattern of hydrogen bonds

assign sstructure rs\_[{ s\_SecondaryStructTypeCharacter | s\_SSstring }]

### Manual assignment of a desired secondary structure annotation to a residue fragment

assign specified secondary structure to the selected residues rs\_, e.g.

```
read pdb "lest"
assign sstructure a_/* "_" # make everything look like a coil
cool a_
assign sstructure a_/1:10 "HHHH_EEEEE"
cool a_
```



This command does not change the geometry of the model, it only formally assigns secondary structure symbols to residues.

Note: to change the conformation of the selected residue fragment, according to a desired secondary string, use the ICM –object and the set command applied to both sequences and molecular objects.

Automated derivation and assignment of secondary structure from atomic coordinates

assign sstructure rs\_

If the secondary structure string is not specified, apply ICM modification of the DSSP algorithm of automatic secondary structure assignment (Kabsch and Sander, 1983) based on the observed pattern of hydrogen bonds in a three dimensional structure.

The DSSP algorithm in its original form overassigns the helical regions. For example, in the structure of T4 lysozyme (PDB code **1031**) DSSP assigns to one helix the whole region a\_93:112 which actually consists of two helices a\_93:105 and a\_108:112 forming a sharp angle of 64 degrees. ICM employs a modified algorithm which patches the above problem of the original DSSP algorithm. Assigned secondary structure types are the following: "H" – alpha helix, "G" – 3/10 helix, "I" – pi helix, "E" – beta strand, "B" – beta–bridge, "\_" or "C" – coil.

Examples:

```
nice "lest"  # notice that many loops look like beta-strands
assign sstructure # now the problem is fixed
cool a_
```

See the set rs\_ s\_SecStructPattern command to actually set new phi, psi angles to a peptide backbone according to the string of secondary structure.

### assign sstructure segment

```
assign sstructure segment [ ms_molecules ]
```

create simplified description of protein topology (referred to as segment representation). Segments shorter than segMinLength are ignored. The current object is the default.

See also show segment, ribbonStyle, display ribbon.

### 2.20.4. break

is one of the ICM flow control statements. It permits a loop (e.g. for or while) to be broken before calculations have completed.

Examples:

```
for i = 1, 8
    print "Now i = ", i, "and it goes up"
    if (i == 4) then
        print "... but at i=4 it breaks, Ouch!"
        break
    endif
endif
```

See also goto.

### 2.20.5. build

The build family of functions allows to create molecular objects

- from sequence file (build s\_seqfile)
- from sequence string (build string)
- from a linear chemical notation (build smiles)
- from a sequence and a template by homology (build model)

It also adds implied hydrogens (build hydrogen) to a molecule and to find a loop in a database (build loop)

### build from sequence file

build [ s\_IcmSeqFileName ] [ library= { s\_libFile | S\_libFiles} ]

reads *s\_IcmSeqFileName.se* ICM-sequence file and builds an ICM molecular object. This sequence file is different from a simple sequence file and contains three (sometimes four) character residue names defined in the icm.res residue library file (try show residue types to see the list). Use macro buildpep if you want to build an object from a string with one letter coded sequences or a named sequence (see the build string command below). The buildpep macro also allows to easily create several molecules by using a semicolon; separator. To get a D-amino acid instead of L-ones simply use D as a prefix: Dala Darg. Specify N- or C-terminal modifiers directly in the file if needed. The build command will create them in some default conformation (extended backbone with different molecules oriented around the origin as a bunch of flowers). Several molecules can be specified in the ICM sequence file.

Residue names can contain numbers (i.e. 4me). However, the residue numbers with a modification character, such as 44a, 44b should contain a slash before the modification character (i.e. 44/a, 44/b). An example in which we create a sequence of residues ala and 4me with numbers 2a and 2b, respectively: "se 2a ala 2b 4me".

Option library= lets you temporarily switch the library file. It can also be done by redefining the LIBRARY.res array of the LIBRARY table.

Examples:

```
build  # def.se file
build "alpha" # alpha.se file
build "wierd" library="mod.res" # get residues from mod.res
#
LIBRARY.res = {"icm","./myres"}
build "a"
```

Use a convenient macro buildpep to build one or several-chain peptides with one- or three- letter code. E.g.

buildpep "ASFGDH;FFF" # two molecules buildpep "ala his trp" # one chain of three residues

### build model by homology

build model seq\_1 seq\_2 ... ms\_Templates ... [ ali\_1 ...] [ margin= { i\_maxLoopLength, i\_maxNterm, i\_maxCterm, i\_expandGaps }

build a comparative model (homology model) of the input sequences based on the similarity to the given molecular objects. The margin arguments:

name	default	description
i_maxLoopLength	999	longer loops are dropped
i_maxNterm	1	the maximal length of the N-terminal model sequence which extends beyond the template
i_maxCterm	1	the maximal length of the C-terminal model sequence which extends beyond the template
i_expandGaps	1	additional widening of the gaps in the alignment. End gaps are not expanded

Possible modes:

• simple one-to-one mode: build model seq\_1 [ms\_1] [ali\_1]

• N sequences – N corresponding molecules: build model seq\_1 seq\_2 .. seq\_N ms\_1,2,..N

Example:

```
l_autoLink = yes
read pdb "x"
read alignment "sx"
build model ly6 a_
ribbonColorStyle = "alignment" # grey-gaps, magenta-insertions
display ribbon
#
read pdb "2ins" # multichain
read sequence unix cat
> a
GIVEQCCASV CSLYQLENYC N
> b
```

```
VNQHLCGSHL VEALYLVCGE RGFFYTPKA
> c
GIVEQCCASV CSLYQLENYC N
> d
VNQHLCGSHL VEALYLVCGE RGFFYTPKA
^D
build model a b c d a_1.
# Now optimize the side chains
selectMinGrad = 1.5
set vrestraint a_/*
montecarlo fast v_/!I/x*
# !I means residues which are not Identical to their template residues
# use refineModel to energetically optimize the model
```

### The algorithm performs the following steps:

**Alignment adjustment:** modifies the alignment according to *i\_expandGaps*, and prepare a sequence with the ends and the long loops truncated according to the alignment and the { *i\_maxLoopLength*, *i\_maxNterm*, *i\_maxCterm* } parameters.

**Building a straight polypeptide from the model sequence:** builds a full-atom polypeptide chain for this new sequence. The residues in your model are numbered according to the template and all the inserted loops residues are indexed with 'a','b', etc. E.g. the numbering may look like this: 200, 201, 203, 204, 204a, 204b, 204c, 205 ... This numbering allows one to follow more easily the correspondence between the template and the model. If you do not like this numbering scheme, just use the

align number a\_/\*

command and the model residues will be renumbered from 1 to the number of residues.

**Backbone topology transfer:** inherits the backbone conformation from the aligned (but not necessarily identical) parts of the known template

Identical side-chain building: inherits conformations of sidechains identical to their template in the alignment

**Non-identical side-chain placement:** assigns the most likely rotamer to the side chains not identical in alignment. If you want to do more than that apply:

```
set vrestraint a_/* ~\# assigns the rotamer probabilities montecarlo fast v_/Cx/x* \# x* selects for all chi (xi) angles
```

You can also manually re-optimize any side chains either interactively (right-mouse click on a residue atom, then select Shake Amino-Acid Side-Chain) or from a script, e.g. for residue 14:

```
set vrestraint a_/* \, # assigns the rotamer probabilities montecarlo v_/14/x* ssearch v_/14/x* \, # systematic conformational search for the 14-th sidechain
```

### Loop searches:

searches the icm.lps which may contain entire PDB-database for suitable loops with matching loop ends and as close loop sequence as possible, inserts them into the model and modifies the side-chains

according to the model sequence.

The loop file can be easily customized, updated and rebuilt with the write model [append] command in a loop over protein structures. To use your custom loop file, redefine the LIBRARY.lps variable.

### Loop refinement and storing alternatives:

adjusts the best loops found and keeps a stack of loop alternatives which can later be tested (see the Homology gui-menu)

### The output

The build model command returns the following variables:

**LoopTable master table** containing list of all the loops, their conformation in alphanumeric code, a measure of the deviation of the database loop ends and the model attachment sites, the loop length and the numerical conformation type (not really important). E.g.

#>T LoopTable							
#>-1_Loop2_Conf3	3_Rmsd	4_Nof	5_Type				
a_ly6.a/7:10 31R21	0.1	11	1				
a_ly6.a/60:63 1RRR32	0.1	8	1				
a_ly6.a/43:46 211331RRRR	0.240658	4	1				

### Individual loop tables

Tables called LOOP1, LOOP2, etc. for each inserted loop. The tables contain the coded conformational string, relative energy, the position of the offset in the structure database file (offset) to be able to extract this loop again, and the rmsd of the loop ends. Example:

```
icm/lv6> LOOP1
#>T LOOP1
#>-Conf-----energy-----offset----rmsd-----
             Ο.
                                      0.092104
  31R21
                          3623594
  31RR2
              1.519275
                          3427772
                                      0.083372
              1.612712
                          3750108
                                      0.097777
  R1121
              1.639177
                          1529882
                                      0.087113
  R1R32
  R1RR2
              1.880638
                          3806768
                                      0.079335
  31R32
              3.714823
                          4561270
                                      0.053853
  R3RR2
              4.531406
                          4003324
                                      0.042881
```

### Writing and restoring the tethers

You can write both template and the model and the tethers between them to files (see section write tethers .

### **Trouble shooting**

build model may crash. A possible reason of the crash is that the pdb file is not correctly parsed due to formatting errors. Many pdb files still have formatting errors, especially those which are generated by other programs or prepared manually. In this case the read pdb command is trying to interpret the field shifts and, as with any guess work, frequently gets it wrong. For example, try 2ins and you will see that the

atom or residue names are shifted. To fix the problem, try to use the exact option of the read pdb command.

### build loop to a model by homology

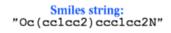
build loop rs\_fragments

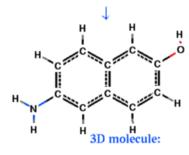
rebuild specified loop based in a PDB-database search (see build model).

An example:

```
read object s_icmhome+"crn"
build loop a_/20:26  # rebuild this loop
```

build[smiles|sln]s\_smiles\_or\_sln[name=s\_ObjName]





create an ICM-object from the smiles-string or sln-string, respectively.

Set l\_readMolArom to no if you do not want to assign aromatic rings from a pattern of single and double bonds (and formal charge and bond symmetrization for CO2, SO2, NO2or3, PO3 ) upon building. To suppress suppress the symmetrization and consequential charging of CO2, set the l\_neutralAcids flag to yes.

### Examples:

build smiles	"CCO" # ethanol
build smiles	"Oc(cclcc2)ccclcc2N"
build smiles	"Oc(cclcc2)cc(ccc3)clc3c2"
build smiles	<pre># dicoronene "clc2ccc3ccc4c5c6c(ccc7c6c(c2c35)c2clclc3c5c6c"+\ "(cl)ccclc6c6c(ccl)ccclccc(c5c6l)cc3c2c7)cc4"</pre>
build smiles	<pre># NAD "[0-]P(=0)(OCC10C(C(0)C10)N1C=2N=CN=C(N)C=2N=C1)"+\ "OP(=0)([0-])OCC10C(C(0)C10)N=1C=CC=C(C=1)C(=0)N"</pre>
build smiles	<pre># Hexabenzo(bc,ef,hi,kl,no,qr)coronene "clc2c3c4c(ccc3)c3c5c(c6c7c(ccc6)c6c8c(ccc6)c6c9"+\ "c(ccc6)c(cc1)c2c1c9c8c7c5c41)ccc3"</pre>
	# rubrene

Usually the build smiles command is not sufficient. The molecule needs to be optimized in the mmff force field and several conformations need to be sampled. The quick way of doing it is shown below.

```
build smiles "CC=1C(=0)C=CC(C=1)=O"
strip a_
build hydrogen
set type mmff
set charge mmff
convert
read library mmff
tolGrad = 0.001
ffMethod = "mmff"
minimize cartesian
```

A more rigorous way of doing the conversion is done with this macro:

```
macro scan2Dto3Dconvert
 oldGrad = tolGrad
 tolGrad = 0.0001
 ffMethod = "mmff"
 vwMethod = "soft"
 unfix V_//a* !V_//avt*
 v = Value(v_//a* !V_//avt*)
 m = Matrix(Nof(v) 2)
 m[?,1] = v
 m[?,2] = Rarray(Nof(v) 170.)
 v = Min (Transpose(m))
 set v_//a* !V_//avt* v
 delete m v
 fix V_//a* !V_//avt*
 delete stack
 store conf 1
 for i=2.5
    load conf 1
   randomize a_// 0.03
   randomize v_//
    store conf i
 endfor
 minimize stack "vw,14,to,hb"
 errorAction = "none" # avoid crashing on cartesian min.
 minimize cartesian stack 3000 "vw,14,to,hb,bb,bs,af"
 if(!Error()) then
    load conf 0
  else
    load conf 1 # try to restore a good conformation
  endif
 vwMethod = "exact"
 minimize cartesian 3000 "vw,14,to,hb,el,bb,bs,af"
 if(Error()) print "ERROR> 2Dto3D failed"
 errorAction = "break"
 delete stack
 tolGrad = oldGrad
 randomize a_//* 0.01
endmacro
```

See also the Smiles function and the find molecule s\_Smiles1 S\_Smiles2 command to find a substructure

### build object from string

build string s\_IcmSequence [ name= s\_ObjName ]

create an ICM-object from a *s\_IcmSequence* string (see the build command above). To get a D-amino acid instead of L-ones simply use D as a prefix: Dala Darg. Specify N- or C-terminal modifiers directly in the file if needed. The build command will create them in some default conformation.

Examples:

```
build string "se nh3+ ala his coo-" name="pep" # one peptide named a_pep.
build string "ml a \nse nh3+ his coo- \nml b \nse trp" # molecules a and b
build string IcmSequence("GHFDSFSDRT","nter","cooh") # translate and add termini
#
# Using alias BS build string "se $0"
BS ala his trp
#
# Using alias BSS build string IcmSequence( $0 )
BSS "SFGDFAGSFG" # quoted one-letter code string
read sequence "GTPA_HUMAN.seq"
BSS GTPA_HUMAN # sequence name
```

### build hydrogens according to topology and formal charges.

```
build hydrogen [ as_heavyAtoms ] [ i_forcedNofHydrogens ]
```

add hydrogens to the specified heavy atoms according to their type and formal charge. All heavy atoms of the current object are used by default. If your have hydrogens already and their configuration is wrong, you can delete them with the delete hydrogen command. The number of hydrogens may be enforced if the optional *i\_forcedNofHydrogens* argument is specified. See also the set bond type command.

Examples:

```
read mol2 s_icmhome+ "ex_mol2" # several small molecules
display a_4.
build hydrogen a_4. # added and displayed
```

### 2.20.6. call icm script

call s\_ScriptFileName [ only ]

invokes and executes an ICM-script file. End the script with the quit command, unless you want to continue to work interactively, or use it in other script.

Option only allows you to suppress opening the script file if the call command is inside a block which is not executed. By default the script file is opened and loaded into the ICM history stack anyway, but the commands from the file are not executed.

### Calling scripts inside conditional expressions.

Examples:

### 2.20.7. center

center[{ as\_| grob } ][ only ][ static ][ margin= r\_margin ]

centers and zooms the screen on selected atoms  $as_{-}$  or graphics objects. Default objects: all existing atoms and graphics objects. The  $r_{-margin}$  argument is given in Angstrom units and can be used to set a relative size of the selection and the frame. Normally all dimensions of the molecule/grob are taken into account, so that the molecule can be rotated without changing scale.

Options:

- only : do not rescale, translate only, i.e. move the selected atoms to the center of the graphics window
- static : scale only according to the visible X-Y dimensions and the margin. Do not take the Z-dimension into account in the size calculation as if you do not intend to rotate objects. That implies an assumption that the orientation of molecules/grobs/maps will not be changed.

Examples:

```
nice "lest"
center
center Sphere ( a_/15:18 )
center a_/1:2 only # keep the scale
read grob s_icmhome+"beethoven" # a genius
display g_beethoven smooth
center g_beethoven static # 10 A margin
```

### 2.20.8. clear

clear terminal screen (no big deal).

### 2.20.9. color family of commands

The color command allows you to color different shell objects, their parts, or different graphical representations with by colors specified in various ways.

### color: main command

### The main color command:

color [ { as\_ | g\_grobName } ] [ wire | hbond | cpk | ball | stick | xstick | surface | skin |
ribbon [base]] [ color\_specification ]

The color can be specified as

 $color_name | s_ColorName | color[i_index] | i_Color | r_Color | I_Color | R_Color | *rgbr=R_3rgb [ window = R_MinMax ]$ 

color selected atoms (*as*\_) or graphics object(s) according to specified color. The window={minValue,maxValue} option allows to provide a range for color mapping. Example:

color ribbon a\_/ Bfactor(a\_/ simple) window=-0.5//2.

This command will clamp Bfactor( $a_/$  simple) values which are normally around zero, but may range from large negative values to large values, to the [-0.5,2.] range.

### The defaults:

- objects: the current object ( a\_ ) only (to color all objects, use a\_\* . )
- graphic representation: wire . To color cpk, xstick, ball, etc., specify them explicitly
- color: the default coloring (atoms by atom type, which can be changed in the icm.clr file; ribbon by secondary structure )

Examples of how the defaults work:

**color\_name.** Color may be just a word (such as black, white, grey, blue, red, yellow, green, orange, magenta, ... ) or string (as "green"), or integer (convenient for automatic coloring within ICM loops), or integer array, or real array (to color according to a certain property, as electric charge or Bfactor). Default: color by atom type; in ribbon representation by secondary structure type.

In DNA and RNA ribbons, bases can be colored separately (e.g. color ribbon base  $a_1/*$  white ), the default coloring being A-red, C-cyan, G-blue, T or U-gold. Colors themselves and all the defaults may be changed in the icm.clr file. Different color specifications with examples:

color red unquoted color name

**color "red"** quoted color name or a string variable

**color color[4]** use color number 4 from the list of **"named"** colors (first section of the icm.clr file). You can show the colors with their numbers by the show color command and their total number is accessible via the Nof(color) function. This mode is useful if you need to color selected elements with contrasting colors rather than with a smooth spectrum.

Example:

```
display a__crn.  # load and display molecule
show colors
color a_/1:5/* color[89]
for i=1,Nof(a_/*)
color a_/$i color[i]  # gay coloring
endfor
```

**color 3 (integer)** color number from the **"rainbow"** section of the icm.clr file. Currently there are 128 colors (i=0,127) in this section and they form a smooth transition from blue to red via white (not really a rainbow). You may change the "rainbow" colors in the icm.clr file. Number 128 becomes blue again. Execute these lines to see how it goes:

```
display "Colors"
for i=1,255
color background i
print i
endfor
```

**color 4.5 (real)** similar to the previous type. The color is interpolated. 4.5 will be the average between the "rainbow color" 4 and "rainbow color" 5.

**color** selection {1,3,5} or color selection {1.,3.,5.} color each element of the selection by I\_Color or R\_Color. The scale is determined by the minimal and the maximal elements of the array, independently of the array length. First the numbers in the array are scaled so that its minimum corresponds to the first color in the "rainbow" section and its maximum to the last color. Then the scaled numbers are applied sequentially to the elements of the selection. If the number of elements in the array is shorter than the number of elements in the selection, the array is applied periodically. In the opposite situation the excessive numbers are not used for coloring but (attention!) they will be used for scaling. Therefore, if you want to influence scaling you may append extremes to the color-property array. See also: GRAPHICS.rainbowBarStyle which determines if and where the color bar will appear.

Example:

**color rgb=**{0.2 0.5 0.1} find the closest color with the following rgb components.

**color rgb="ff0000"** string representation of rgb color (like in html), each of the three colors (red,green,blue) is defined by **two** characters in hexadecimal form.

### color background

color background [ *Color* | *s\_Color* | *i\_Color* ] colors the background to the specified color.

Examples:

```
buildpep "ASDWER"
                      # hexapeptide
color a_/1:4 green
                      # the first four residues in green
color
                      # return to default colors by atom type
                      # load, display crambin from the PDB file 1crn
display a__1crn. only
                      # color atoms according to their B-factor
color a_1crn.//* Bfactor(a_1crn.//*)
                      # crambin's ribbon
                      # from blue N-term to red C-term gradually
display a_/* ribbon only
color a_/* Rarray(Count(1 Nof(a_/* ))) ribbon
                      # another crambin's ribbon
                      # from blue N-term to red C-term gradually
color background blue
                      # thick worm representation
assign sstructure a_/* "_'
GRAPHICS.wormRadius= 0.9
display a_/* ribbon only
color a_/* Count(1 Nof(a_/* )) ribbon
```

### color cursor

```
color cursor [ colorName ]
```

colors the residue cursor which can be displayed with the display cursor *rs*\_[*colorName*] command.

Example:

```
nice "lcrn" display cursor a_{\rm }/21~ # use arrows to move the res. cursor color cursor blue
```

See also: display cursor, Res( cursor )

### color grob : special ways to color grobs

### automatic assignment of different colors to different grobs

color grob unique

In addition to the main color command which colors grobs there is a special command to automatically assign the displayed grobs to different colors. See example for the split command.

### color grob by matrix of RGB values for each vertex.

color g\_grob M\_rgbMatrix

a special command to color grobs by colors defined for every vertex by three RGB numbers. This type of matrix is returned by the  $Color(g_grob)$  command. This command allows for gradual disappearance of a grob into background.

Example:

### color grob by map: coloring surfaces by 3D scalar field

color grob { map | map\_VolumeProperty } I\_transferFunction R\_bounds

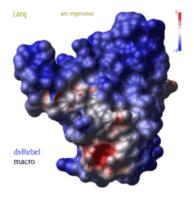
colors the surfaces of grobs by the interpolated **nearest** values of scalar field defined by the *map\_VolumeProperty* map.

Transfer function is the same as in color map.

### color grob potential : coloring grob by electrostatic potential

color grob potential [ fast ] ms\_sourceAtoms

(REBEL feature) calculates electrostatic potential waterRadius away from the surface of the  $g\_skin$  graphics object and color surface elements according to this potential from red to blue. The potential is calculated either by the REBEL boundary element solution of the Poisson equation, or, if option fast is specified, by a simple Coulomb formula with the dielConstExtern dielectric constant (78.5 by default).



The local value of potential is clamped to the range [ -maxColorPotential, +maxColorPotential]. It means that a potential larger than maxColorPotential is represented

by the same blue color, while values smaller than maxColorPotential are represented by the same red color. The real range is reported by the command and you can adjust maxColorPotential to cover the whole range. To suppress the absolute maxColorPotential threshold and use auto-scaling instead set maxColorPotential to 0. The color bar with values will appear according to the GRAPHICS.rainbowBarStyle preference. There are two macros to generate potential-colored skins: rebel and rebelAllAtom

The second one (given below) considers all the atoms (including hydrogens) with their charges.

Examples:

```
macro rebelAllAtom ms_
display ms_
make grob skin ms_ ms_
make boundary ms_
color grob potential ms_ # HERE
display grob smooth
endmacro
read object "crn"
rebelAllAtom a_1
```

See also: electroMethod, make boundary, delete boundary, show energy "el", Potential().

### color label

color label [ *as*\_] [ *Color* | *s\_Color* | color [ *integer* ] | *i\_Color* | *r\_Color* | *I\_Color* | *R\_Color* ]

Color labels associated with the selected residues or atoms.

Examples:

```
read object "crn"
display a_//n,ca,c white
display label residue
color label a_/* Count(1 Nof(a_/*))
color label a_/5:10 magenta
```

See also: display label, resLabelStyle.

### color map

color map [ s\_mapName ] [ I\_colorTransferFunction ]

color the current or the specified map according to the color transfer function supplied as  $I\_colorTransferFunction$ . All grid points are divided to Nof( $I\_colorTransferFunction$ ) color classes according to the normalized function value (sigma units around the mean value) and each class is colored as specified in the  $I\_colorTransferFunction$  (0 means transparent).

If the number of *I\_colorTransferFunction* elements is odd  $(2^* n+1)$  the class boundaries are the following:

- –infinity
- Mean– *n* \*sigma,
- Mean–(n-1)\*sigma,
- Mean–( *n* –2)\*sigma,
- ...
- Mean- 1\*sigma,
- Mean
- Mean+ 1\*sigma,
- ...
- Mean+(*n*-1)\*sigma,
- Mean+(n)\*sigma.
- +infinity

For even number of elements  $(2^* n)$ , boundaries are shifted by half a sigma, so that the middle class is between Mean-0.5\*sigma and Mean+0.5\*sigma. Color codes are in arbitrary units since the array is normalized so that the highest value corresponds to the red color. Deep blue is 1. Zero is always the transparent color (no coloring). The spectrum is defined in the icm.clr file. Examples of coloring:

- {0 0 0 0 0,0 0 0 3 10} default map coloring, color only high densities (blue from 3 to 4 Sigma, red >4 Sigma). Comma only shows you where the mean is.
- $\{0 \ 1 \ 0\}$  color only Mean+- 0.5\*sigma nodes, ignore high and low densities.
- $\{1 \ 0 \ 2\}$  color low and high densities by different colors, ignore densities around the mean.
- {1 2 3 0 5 6 7} similar the previous one, but with more grades

### Example:

color map  $\{1 \ 2 \ 0 \ 4 \ 5\}$ 

### color molecule

```
color molecule [ ms_molecules ]
```

a special command to color the displayed and selected molecules differently. It is a bit tricky (solely for your convenience): if there are both proteins and small "hetero"-molecules (e.g. waters), the colors will not be wasted on little guys. If all the molecules are non-protein (i.e. no residues defined as amino-acids, or  $a_A$  is empty ), than they will be colored differently.

### color ribbon

color ribbon

color the displayed ribbon. See also base, GRAPHICS.dnaBallRadius, GRAPHICS.dnaStickRadius, and other DNA settings in GRAPHICS.

### color volume

```
color volume [ Color | s_Color | i_Color ]
```

determines the color of the fog in the depthcueing mode ( activated with Ctrl-D). For example, if you want that distant parts of you structure are darker (black fog), but the background is sky-blue, you will do

the following:

Examples:

color background blue color volume black

# 2.20.10. compare: setting conformation comparison parameters for the montecarlo command

compare { as\_| vs\_ } options

The goal of the two following compare commands is to provide a desired **setting before the montecarlo command**. This command defines a filter which is used to decide how many and what conformations from the stochastic optimization trajectory are kept as low energy representatives of a certain area in conformational space. This metric is also used for the subsequent **stack** manipulations, e.g. compress stack.

The compare command defines the distance measure between molecular conformations which is used to form a set of different low energy conformers in the course of the stochastic global optimization procedure. The defined distance is compared with the vicinity parameter and determines whether two conformations should be considered different or similar (i.e. belonging to the same slot in the conformational stack). The compare command determines the spectrum of conformations that will be retained in the stack, accumulated during a montecarlo procedure. The default comparison set is a set of all free torsion variables (see compare vs\_). Other methods compare atom RMSD with and without superposition, and compare only the atoms in the vicinity of a static object ( compare surface ).

### compare by deviations of cartesian coordinates

compare[static]as\_

The command needs to be run when Cartesian root-mean-square deviation for positions of selected atoms  $(as_)$  as a distance measure between stack conformations. Set the vicinity parameter to about 2.0 Angstrom if you want to consider conformations deviating by more than 2 A as different conformational families.

By default the selected atoms in different conformations will be optimally *superimposed before* the coordinate RMSD is calculated. The static option suppresses superposition and measures absolute deviation of the coordinates between conformations. The static option is relevant for ligand atoms in docking simulations to a static receptor.

The result of this procedure is that an internal flag is set to perform cartesian RMSD calculations during montecarlo run, and a set of selected atoms is marked for comparison.

### compare by deviations of internal coordinates/torsions.

compare vs\_

use angular root-mean-square deviation for selected internal variables (usually torsion angles) as distance (set vicinity to at least 30.0 degrees accordingly)

Examples:

<pre>compare v_//phi,psi vicinity=30.0</pre>	#	compare ONLY the backbone angles consider two conformations with phi-psi RMSD < 30. as similar
compare a_2//ca static	#	compare Cartesian deviations of the second molecule's alpha-carbon atoms
vicinity=3.0	#	without prior optimal superposition consider two conformations with second molecule deviation < 3 A as similar

### compare surface: dynamically selecting comparison atoms

compare surface *as\_currentObjSelection* | as\_staticReferenceObject.

dynamically calculates a subset of *as\_currentObjSelection* **near** *as\_staticReferenceObject* for comparison by static RMSD inside montecarlo command.

This is useful for protein-protein docking simulations when you want to measure the sRmsd distance between the current conformation and the stack conformations ONLY for the interface residues of the moving molecule. The interface residues are dynamically determined as those which are close to the static receptor specified in the second part of the selection. This static receptor should reside in a separate object.

The *vicinity* size is determined by the selectSphereRadius parameter

An example in which we sRmsd-compare only those carbons of barstar which are next to the barnase surface.

```
read pdb "lbgs"  # a complex
read pdb "lal9.a/"  # the protein ligand only
convert
...  # make maps and other actions to prepare protein-protein docking
compare a_//c* | a_1.1 surface  # will use only
selectSphereRadius = 7.
...
montecarlo
```

### 2.20.11. compress

compress stack [ i\_fromConfNumber i\_toConfNumber ]

Remove similar and/or high energy conformations from the conformational stack. During a montecarlo run, some conformations of the generated conformational stack may be substituted by newly calculated ones with lower energies. New conformations may violate the initially correct distribution of the conformations in the slots of the stack as defined by the vicinity parameter and by comparison mode specified by the compare command. The **compress** command compares all the pairs of the stack conformations, identifies pairs of conformations in which two conformations are separated by a distance less than the vicinity threshold, and removes the higher energy stack conformation from each close

2.20.10. compare: setting conformation comparison parameters for themontecarlo command

pair. Optional arguments *i\_fromConfNumber* and *i\_toConfNumber* define a subset of the conformations in the stack which are to be analyzed and compressed (if any). The whole stack (from the first to the last conformations) is processed by default.

Note that if two close conformations are compressed into the better energy one, the number of visits of the resulting conformation will be a **sum** of the two numbers of visits.

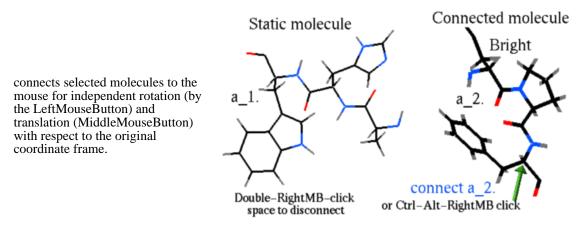
See also How to merge and compress several conformational stacks

Example (define a distance and compress) we generate two stacks, merge them and re-compress two sets with a different comparison criterion:

```
buildpep "VTLFVALY"
mncallsMC = 5000
montecarlo  # generates stack, compar
write stack "f1"
delete stack # clean up and
montecarlo  # generates another stack
read stack append "f1"  #
compare v_/2:5/phi,psi  # compare settings are different
vicinity = 40.  # new vicinity
compress stack
```

### 2.20.12. connect

connect [ append ] [ none ] [ { ms\_molecule | grob } ]



Option append will add selected molecule to the previously connected molecules

Note, that rotations/translations in the connect mode actually **change the atomic coordinates** of the selected molecules and keep the coordinate system unchanged in your graphics window.

To restore the usual global mode (i.e. all objects/molecules are disconnected and the mouse does not change their absolute positions, but rather the point of view), hit the Esc key when the cursor is in the

graphics window. To restore the global mode **temporarily** press the Shift button. Connect can also be activated by a --Ctrl-Alt-RightMB-Click on any atom of the chosen molecule (see graphics controls).

Use: connect none to switch back to the global connection

### 2.20.13. continue

continue

skip commands until the nearest endfor or endwhile. Note, that in contrast to FORTRAN (god bless it), it is NOT a dummy operator.

Example:

```
for i=1,10
    if i==3 continue # do not print 3
    print i
endfor
```

See also: flow control statements.

### 2.20.14. convert

convert [ exact ] [ charge ] [ tether ] [ os\_non-ICM-object ] [ s\_newObjectName ]

converts an incomplete non-ICM-object (e.g. object of type 'X-Ray' resulting from the read pdb command) into a true ICM-object for which you may calculate energy, build a molecular surface and perform all operations.

There are two principally different modes of conversion. In the default mode the program looks at the **residue name** and tries to find a full-atom description of this residue in the icm.res file. This search is suppressed with the exact option.

Hydrogen atoms will be added if the converted residues are known to the program and described in the icm.res library. If the object selection is omitted, the current object will be converted. If default  $s_{newObjectName}$  is generated by adding number "1" to the source object name.

The default convert command is best used to convert PDB entries which have explicit residue descriptions and usually do not have hydrogen coordinates. In this mode each residue name is searched in the icm.res file and the coordinates of the present heavy atoms are used to calculate the internal geometrical variables (bond lengths, bond angles, phase and torsion angles) for the full atom model.

### The exact option: converting protein with unusual amino-acids

Some pdb-entries may contain non-amino acid residues, or *modified* amino-acid residues which do not need to be replaced by standard full atom library entries with *the same name*. In this case use the exact option. This option suppresses interpretation by short residue name and converts the **existing** atoms and bonds in single-residue molecules (amino acids in peptides and proteins will still be extended by hydrogens upon conversion, to suppress that conversion write the molecule as mol and read it back, then

convert exact ). Option exact may be necessary because chemical compounds with a four-letter short name identical to one of the amino-acid residues, could be mistakenly converted into an amino-acid with a corresponding name.

### The charge option

Normally, upon convertion, the atomic charges are taken from the icm.res library entries. Option charge tells the program to inherit atomic charges from the *os\_non-ICM-object*. For small molecules, use set charge, set bond type and, possibly, build hydrogens before conversion of a new compound. i\_out will contain the number of heavy atoms missing from the pdb-template.

### Additional cleanup

Actually more precedures need to be performed to prepare a functional object from crystallographic coordinates, e.g. identifying optimal positions of added polar hydrogens, assigning the most isomeric form of histidine, etc.

We recommend the convertObject macro instead of the plain convert command to achieve those goals.

### **Refining the model**

To refine a model use the refineModel macro.

#### convertObject macro

The convertObject macro is a convenient next layer on the convert command. The macro may convert only a few molecules out of your pdb file, optimize hydrogens and do some other useful improvements of the model.

### Comparing convert, minimize tether and regularization.

It is important to understand the difference between the **convert** command, the minimize tether command and the regularization procedure implemented in the macro regul.

All three create ICM-objects from PDB coordinates, but details of generated conformations and the amount of energy strain will differ.

We recommend to use convertObject macro for most serious applications involving energy optimization.

#### convert

- uses all-atom residue templates (including hydrogens) from the icm.res library
- creates temporary ICM-library descriptions for unknown residues
- makes geometry **identical** to the PDB coordinates: bond length and bond angles may be distorted.
- the converted structure will be energy strained because of common imperfections of the PDB entries and the hydrogen atoms added by the procedure
- C-alpha-only structures will not be properly converted because a special prediction algorithm is required to extrapolate the coordinates of all atoms from C-alpha atom positions.

- these objects are good enough for graphics, skin, secondary structure assignment, rigid body docking. They are not good for loop modeling and side-chain modeling.
- needs to be followed by polar hydrogen placement and histidine state prediction ( implemented in the convertObject macro )

## minimize tether threading a regular polypeptide through an incomplete/gapped set of coordinates.

- you need to create a sequence file first and use the build command;
- you will need to create the missing residues manually, say, with the write library command;
- build will use all-atom residue templates including hydrogens, and will preserves the fixation;
- the linear chain with *fixed idealized covalent geometry* or, actually, any fixation you define, will be threaded onto the PDB coordinates in the best possible way;
- Ca-atom PDB structures will be handled properly if all backbone torsion angles are unfixed;
- the resulting ICM-object will be strained and will need further relaxation.

#### full regularization and refinement

- uses minimize tether to create the starting conformation;
- employs a multistep energy minimization (annealing) of the structure to relief energy strain;
- these are the best objects that can create in ICM for further simulations.

(see macro regul for details).

Examples:

#### Converting a chemical compound from a mol/sdf or mol2 files.

To convert a chemical from GUI menus, follow these steps:

- make sure that bond types and formal charges are correct
- select the MolMechanics.ICM-Convert.Chemical menu item, check the parameters and press OK. Normally to convert from 2D to 3D you need to optimize the ligand. ICM will perform a multiple start global optimization using the MMFF94 force field (internally it runs the convert2Dto3D macro). If you want to preserve the geometry, select the keepGeometry option.

**Command line conversion** To perform the same conversion in a *batch* run the convert2Dto3D macro, or, to make a conversion without full optimization from a command line or script, issue the following commands:

```
# assuming that bond types and formal charges are correct
build hydrogen
set type mmff
set charge mmff
randomize a_//!vt* 0.01 # sometimes it helps to avoid singularities
convert
set v_//T3 180. # making flat peptide bonds
fix v_//T3 # optional
```

### Converting a chemical compound and rerooting the tree at the same time

```
convert as_rootAtom
```

if an atom selection is provided instead of the object selection, the tree will be rerooted to the selected atom.

### 2.20.15. copy

copies stuff which CANNOT be copied by direct assignment such as: a=b

```
copy[strip][tether]os_[s_newObjectName][delete]
```

create a copy of  $os_{\rm with}$  the specified name. Default source object is the current object. The default name is "copy" (object a\_copy.)

Option delete (must be specified at the end of the line) forces the command to overwrite the object with the same name if there is a name conflict.

Option strip applies the strip operation to the copied object. The stripped object has a PDB type and is much smaller in memory.

Option tether applies tethers from the source object to the atoms of the copy-object. For further refinement see the refineModel macro.

Examples:

### 2.20.16. crypt

crypt key= s\_password { s\_fileName | string= s\_string }

encrypts the file *s\_fileName* or string *s\_string* in place (the size of the encrypted file/string is exactly the same), adds extension .e to the file name. If string is encrypted, its name is not changed. Apply the operation again to restore the file or string. You may encrypt both text and binary files. Note that this command has nothing to do with the unix crypt utility. ICM uses different algorithm.

Example:

crypt key="HeyMan" "\_secretScript" # encrypt and create \*.e file crypt key="HeyMan" "\_secretScript.e" # decrypt it ss="Secret rumour: Div(Rot(F))=0 !" crypt key="fomka" string = ss # encrypt show ss crypt key="fomka" string = ss # decrypt show ss

### 2.20.17. delete ICM shell objects

delete shell objects or their parts.

#### delete ICM-shell object

```
delete[alias][alignment][factor][grob][iarray][integer][logical][
macro][map][matrix][profile][rarray][sarray][sequence][string]{ name1 |
s_namePattern1 } name2 ...
```

ICM-shell objects have unique names; to delete some of them just type

delete { icm-shell-objectName1 | s\_namePattern1 } icm-shell-objectName2 ...

You may use name patterns with **wildcards** (see pattern matching) and add explicit specification of the ICM-shell object type, if you want the search to match only the objects of particular type. If the ICM-shell object type is not specified, all the shell-variables will be considered.

Examples:

```
delete aaa  # delete ICM-shell object aaa
delete a b c  # delete ICM-shell objects a, b and c
delete "*"  # delete ALL ICM-shell objects added by user
delete "mc?a*"  # delete ICM-shell objects matching the pattern
delete rarrays  # delete ALL real array
delete objects  # delete ALL molecular objects, same as delete a_*.
delete rarray "a*"  # delete real arrays starting with 'a'
```

#### delete alias

delete alias

see alias delete alias\_name. Example:

```
alias ls list
alias delete ls
```

2.20.17. delete ICM shell objects

### delete atom or variable selection

delete as\_selectionName

or

```
delete vs_selectionName
```

delete atom or variable selections. **Important:** keep in mind that deleting the named selection is not the same as deleting actual molecules or objects selected by them (see below: delete os\_or delete ms\_). The number of named selections is limited to about 10 in each category, therefore you may need to delete them from time to time.

Examples:

### delete atom

delete *as\_atoms* : \*delete \*atoms as\_namedSelection

delete selected atoms in a non-ICM object. The selection here must be a constant atom selection, rather than a named selection (e.g. you can say delete  $a_{1:10/*}$  but NOT aaa =  $a_{1:10/*}$ , delete aaa).

To delete a named variable, use delete atom name Example:

```
read pdb "1crn"
delete a_/1:10/*
aaa = a_/18:20
delete atom aaa
```

### delete hydrogen

delete hydrogen as\_

delete selected hydrogen atoms in a non-ICM object. See also build hydrogen. To delete hydrogens in an ICM object, strip it first.

### delete object

```
delete { object | os_ }
```

delete molecular object. Make sure that you specify an object selection (a\_lcrn. is correct, a\_lcrn.\* or a\_lcrn.//\* is INCORRECT.) To delete an object from a selection variable (as\_out,as2\_out or

as\_graph, or any use defined aselection variable), use delete atom *as\_namedSelection* (e.g. delete atom as\_graph) or specify the selection level explicitly.

Examples:

```
delete object  # delete ALL molecular objects
delete a_*.  # delete ALL molecular objects
delete a_2,4.  # delete objects number 2 and 4
delete a_2a*.  # delete objects with names starting from 2a
read pdb "lcrn"  # load crambin
convert  # create the second object named lcrn_icm
  # from the pdb object
delete a_1.  # delete the 1st pdb-object
delete Object( as_graph )  # graphical selection
```

#### delete molecule

delete [molecule] ms\_

delete separate molecules from molecular objects. The integer reference number(s) of molecule(s) which can be shown by the show molecule command and used in molecule selections are redefined after deleting or moving molecules from or in the ICM-tree, respectively.

To delete a molecule from a selection variable (as\_out,as2\_out or as\_graph, or any use defined aselection variable), use delete atoms  $as_namedSelection$  (e.g. delete atom as\_graph) for non-ICM objects, or use the Mol function to specify the selection level explicitly (e.g. Mol( as\_graph )).

Examples:

```
read pdb "2ins"  # load insulin with water molecules
delete a_2ins.w*  # delete water molecules
delete atoms as_graph  # deletes selected non-ICM atoms/molecules
delete Mol( as_graph )  # deletes selected non-ICM atoms/molecules
```

### delete bond

delete bond as\_singleAtom1 as\_singleAtom2

delete a covalent bond between two selected atoms. This command is used to correct erroneous connectivity guessed by the read pdb command. It is particularly important when you are going to create a new ICM-residue using the write library command and the entry to it in the icm.res or your own residue file (it has the same format). In interactive graphics mode you may type delete bond and then click two atoms with the CTRL button pressed.

Examples:

See also: make bond and make bond atom\_chain.

### delete boundary

delete boundary

an auxiliary command to free additional memory allocated by the make boundary command.

### delete conf

```
delete conf i_stackConfNumber [ i_stackConfNumberTo ]
```

delete a specified conformation from the stack or a series of conformations starting from *i\_stackConfNumber* to *i\_stackConfNumberTo* 

### delete drestraint

delete drestraint [ *as\_1* [ *as\_2* ] ]

delete distance restraints formed between specified atom selections *as\_1* and *as\_2*. If no selection is specified all distance restraints are deleted

Examples:

delete drestraint a\_mol1 a\_mol2 # intermolecular restraints

### delete label

delete label i\_StringLabelNumber

delete graphics string label (text in the graphics window). These strings have no unique identification names, they are just numbered. Numbers are compressed as you delete some labels from the middle of the list.

Examples:

delete label 1 # delete the first displayed label

See also:

show label to find out the label number and display label to create and display a string label.

### delete sequence

delete sequence *i\_NofLastSequences* 

delete sequence [ i\_minLength i\_maxLength ]

- no arguments: delete all ICM-sequences
- one integer argument: delete last i\_NofLastSequences sequences

• two integer argument: delete sequences shorter than *i\_minLength* or longer *i\_maxLength* 

### delete site

delete site [ { s\_siteString | i\_siteNumber| I\_siteNumbers } ] ms\_

delete the sites of the selected molecules. The sites can be specified by their name, or number. All sites are deleted by default.

Example:

```
nice "lest"  # has 13 sites.
delete site a_1.1 12
delete site a_1.1 {3,4,5}
delete site a_1.1 "FT CONFLICT 178 178" # blanks are unimportant
delete sites  # delete all of them
```

### delete sstructure

delete sstructure seq\_

delete the assigned secondary structure to prepare the sequence for the secondary structure prediction (see the Sstructure function).

### delete disulfide bond

delete disulfide bond [ all ] [ { rs\_Cys1 rs\_Cys2 | as\_atomSg1 as\_atomSg2 }]

delete specified or all disulfide bridges in ICM objects.

Examples:

See also: make disulfide bond and (important!) disulfide bond.

### delete peptide bond

```
delete peptide bond [ as_N as_C ]
```

delete specified extra peptide bonds in ICM objects (e.g. imposed to form a cyclic peptide).

Example:

```
delete peptide bond a_/15/c a_/29/n
```

See also: make peptide bond and peptide bond.

### delete stack

delete stack

delete stack of conformations.

See also read stack, write stack, and delete conf.

### delete table

delete { T\_table | table\_expression }

delete the specified complete table or just the entries selected by the expression.

Examples:

group table t {1 2 3} "a" {4. 5. 7.} "b"
delete t.a == 2 # the second entry
show t
delete t # the whole thing

### delete term

delete term *s\_terms* 

switch off the specified terms of the energy/penalty function.

Examples:

delete terms "tz,sf" # do not consider tethers and solvation contributions

### delete tether

delete tether [ as\_]

delete tethers of the specified atoms (as\_), if no selection is specified all tethers are deleted.

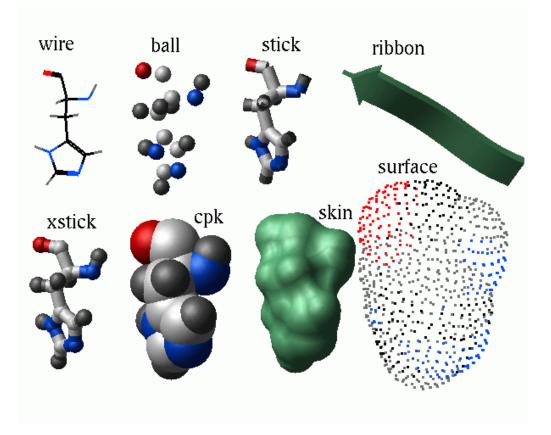
### 2.20.18. display

display molecules or graphical objects

### display model

display [wire |cpk |ball |stick |xstick |surface |skin |ribbon [base ]] [as\_ [as\_2]] [ color ] [virtual ] [plane ] [center [center\_options]]

display specified graphics primitives for selected atoms or residues.



Once something is displayed and your cursor is in the graphics window you may rotate, translate, zoom and move both clipping planes with the mouse and keystrokes.

To refer to the base part of DNA/RNA represented as ribbon, use the additional specifier called **base**, which can be separately displayed and colored. E.g.

```
makeDnaRna "ACTG" "mydna" yes yes "dna"
display ribbon base
color ribbon base a_1 blue
```

Display surface atoms may be defined by TWO arbitrary selections (it would mean: display surface of atoms  $as_1$  as they are surrounded by atoms  $as_2$ )

Defaults: wire representation, all atoms, coloring according to atom type.

color options

The color can be specified by a number of ways (see the color command for a more detailed description) : *Color* (e.g. red), s\_Color (e.g. "red"), numerical color:

*i\_Color* | *r\_Color* | *I\_Color* | *R\_Color* [ window= *R\_2minmax* ]

The window array of min and max values allows to clamp the value you want to map to a color to the specified range.

#### **Other options:**

**center** : will perform the center command on the displayed object(s).

plane : will display the object the way it was saved, e.g.

Example:

```
nice "lcrn"
# Display something and save the object with the graphical information
color ribbon green
display a_/3:5 cpk
write object auto "tm"
q
% icm
read object "tm" # now contains graphics information
display plane
```

virtual : additionally displays the coordinate axes, virtual atoms and virtual bonds starting from the origin. It is a good way to visualize the whole ICM molecular tree as it grows from the origin. This option is applicable only to the ICM molecular objects.

More examples:

```
buildpep "AFSGDH;OWRTEY"
                               # two peptides
                               # display current object and color atoms
display
                               # according to atom type
display a_1 red
                               # display the first molecule and color it red
display skin a_/5 a_* yellow
                               # display skin of the 5th residue
                               # as surrounded by all the atoms
display ribbon
                               # display ribbon for all the residues
read pdb "2drp"
                               # a pdb file
assign sstructure a_1/123:134,153:165 "H"
                                                 # No sstructure in 2drp
assign sstructure a_1/109:114,117:121,141:144,147:151 "E"
display a_1 ribbon red
                                                # two Zn-fingers
display a_1/113,116,143,146/!n,c,o xstick blue
                                                # Cvs residues
display a_1/129,134,159,164/!n,c,o xstick navy # His residues
display a_2,3 cpk magenta
                                                # Zn-atoms
adna1=a_4//p,c3[*],c4[*],c5[*],o3[*],o5[*]
                                                # two DNA chains
adna2=a_5//p,c3[*],c4[*],c5[*],o3[*],o5[*]
display adnal xstick white
display adna2 xstick aquamarine
display adnal adnal surface white
display adna2 adna2 surface aquamarine
center
display "Zn-finger peptides complexed with DNA" pink
```

# display 4 chains of insulin as 4 thick worms colored from N-to C-terminus read pdb "2ins" color background blue assign sstructure a\_/\* "\_" # thick worm representation GRAPHICS.wormRadius= 0.9 display a /\* ribbon only color a\_1/\* Count(1 Nof(a\_1/\* )) ribbon color a\_2/\* Count(1 Nof(a\_2/\* )) ribbon color a\_3/\* Count(1 Nof(a\_3/\* )) ribbon color a\_4/\* Count(1 Nof(a\_4/\* )) ribbon # examples of DNA and RNA ribbons nice "4tna" resLabelStyle = "A" display residue label color residue label a\_/??u gold # ??u also selects modified Us color residue label a\_/??a red

### display new: refresh or unclip view

display new: \*display \*restore display restore plane

commands to mimic some of the interactive controls. These commands are primarily used in GUI commands (see icm.gui file) and scripts/macros.

**new**: rebuilds some graphical representations (e.g. your as\_graph has been changed in the shell and you need to refresh the image, or you changed the orientation and want to redisplay the labels elevated above the skin surface by resLabelShift).

restore : a softer action than new.

restore plane : moves the clipping planes beyond the displayed objects (keystroke: Ctrl-U, or the 'Unclip' button).

### display off-screen

```
display off [ i_Width i_Height ]
```

Sometimes you want to generate some images in a script **without** opening an explicit graphics window. The display off command opens an off-screen rendering buffer of *i\_Width* by *i\_Height* size in pixels, in which all the usual display/color/undisplay/center commands work as usual. **NOTE:** one cannot have both off-screen and on-screen displays in one ICM session.

An example script (can also be performed interactively):

```
display off 400 300
nice "lest"
rotate view Rot( {0. 1. 1.} 50.)
write image "est1"
unix xv est1.tif
set window 700 800 # NB: 'center all' will be applied
write image "est2"
unix xv est2.tif
```

```
display a_/4/o cpk
center a_/3,4
write image "est3" rgb
unix xv est3.rgb
build string "se ala trp"
display off 400 300
display skin
write image "est3" rgb delete
unix xv est3.rgb
```

### display origin

display the axis of the coordinate frame. The length of the arrows is defined by the axisLength parameter. Use undisplay origin to undisplay it. E.g.

```
read pdb "lcrn"
display
display origin
undisplay origin
```

### display box

display box [R\_6boxCorners]

display graphics box specified by x,y,z coordinates of two opposite corners of a parallelepiped. This box can be resized and translated interactively with the Left and Middle mouse buttons:

- Resizing: Grab a corner of the box with the Left-Mouse-Button and drag it to resize the box
- Translating: Grab a corner or a center of the box with the Middle-Mouse-Button and translate

Example:

```
buildpep "ala his trp" display box Box(a_1) # change it interactively
```

See also the Box () function which returns six parameters describing the box.

Examples:

### display cursor

```
display cursor [ alignment ] [ rs_ ] [ color ] [ s_Symbol ]
```

display the residue cursor at the specified single residue  $rs_{-}$ . Move the cursor with the left and the right arrows. You may redefine the color (default 'red') and the cursor symbol (default '#').

If the alignment option is specified, a pairwise alignment of the cursor molecule (the model) and the template to which residues of this molecule are tethered is shown. To get the alignment you need two objects:

- an ICM-object of your model, it can be just an extended polypeptide after the build command, or a preliminary model. Set your cursor to this molecule
- the template (a simpler non-ICM object is possible) to which the residues of the model are tethered with the set tether command.

A set of commands for editing the alignment includes: Arrows, Insert and Delete for aligning and unaligning the cursor residue with its template counterpart, respectively, Ctrl-arrows for shifting the tethers; PageUp and PageDown for 'undo' and 'redo' actions, respectively. It is important that the alignment is a derivative of the tether pattern, and what you really edit is tethers.

Example:

## display clash

display clash [ as\_1 [ as\_2 ] ] [ r\_vwReductionRatio ]

display all the interatomic distances between two atom selections which are shorter than the sum of van der Waals radii multiplied by the  $r_vwReductionRatio$  parameter (0.8 by default). IMPORTANT: this will work only for the ICM-objects. Use the show energy "vw" command (and pay attention to the current fixation) to precalculate interaction lists.

This command shows many irrelevant short contacts. dsEnergyStrain, display gradient, etc. seem to be more informative.

See also: show clash, undisplay and atom energy gradient (force) analysis with: show a\_//G or display a\_//G.

Example:

```
read object "crn"
show energy "vw"
display clash a_/11 a_/!11 0.75 # distances < (R1+R2)*0.75
# this is an alternative method which analyzes the gradient
selectMinGrad = 100. # analyzes forces greater than 100
display ribbon grey
display Res( a_//G )
display gradient a_//G
color Res( a_//G ) ribbon magenta</pre>
```

## display drestraint

```
display drestraint as_
```

displays drestraints, disulfide bonds, and peptide bonds imposed on selected atoms.

See also: read drestraint, set drestraint, make disulfide bond, make peptide bond, make drestraint.

#### Example:

```
build string "se ala his trp"
display
set drestraint a_def.al/3/hz2 a_def.al/1/hb3 2
display drestraint
minimize "vw,14,to,cn"
```

## display gradient

display gradient as\_

display vectors of energy derivative with respect to atom positions or selected atoms as\_.

**Important**: the gradient must be **pre-calculated** by using one of the following commands: show energy or minimize. The values of gradient components (lengths of vectors for each atom) can be shown by show gradient *as\_*. When a gradient vector is displayed, two transformations are performed: it is scaled and colored to represent the range of values in the most convenient and natural way while still being able to deal with a wide range of gradient values from negligible to 10 to the thirtieth power, as may be the case for a strong van der Waals clash. When all gradient vectors are under 20 kcal/mole\*A they will be colored by the "cold" colors (blue...green...yellow) and will be assigned a length less than 2 Angstroms. If you see a red and long vector you may have a problem. Check it by zooming in and using show gradient *as\_*. You can also select only atoms with gradient greater than the threshold value selectMinGrad by typing a\_//G and display only specified strained atoms. It helps to get rid of little blue arrows for unstrained atoms.

Examples:

```
buildpep "ala his trp glu leu"
randomize v_//phi,psi
show energy
selectMinGrad= 200.
display gradient a_//G
```

## display grob

display grob [ solid ][ smooth ][ dot ][ reverse ] [ transparent ]

display g\_Name1 g\_Name2 ... options

display grob select ... options

display all, specified, or graphically selected graphics object(s). They are referred to as **grob** in the ICM-shell and as "3D meshes" in the GUI interface. The display grobs command will display all existing graphics objects. Options:

- dot will show only dot-vertices of the object.
- reverse to invert lighting; this option will change directions of the grob surface normals (will turn the grob inside-out)
- smooth enforces the Gouraud shading method to smooth the solid surface.
- solid allows solid surface representation of the object and requires that the original object has information about triangles forming the solid surface.
- transparent makes solid grob transparent

One can also color and undisplay graphics objects, as well as connect to them.

Examples:

```
read matrix "def.mat" # 2D sin(r^2)/r^2 function of a grid
make grob solid def # convert matrix into a graphics object g_def
display g_def smooth # a hat of the 22st century
display g_def reverse # shine light from inside the head
display grob transparent # like Lenin in Mausoleum
# now you can double-LeftMB-click on it and Alt-X it to your satisfaction
```

## display hbond

```
display hbond [ as_ ] [ r_maxHbondDistance ] [ only ]
```

Only hydrogen bonds of the current object may be displayed. Before calling this command, you should use any of the following commands: show hbond, show energy, minimize to calculate the list of hydrogen bonds. The real argument *r\_maxHbondDistance* defines an upper bound of the distance between a hydrogen and a potential hydrogen acceptor to place the pair to the hydrogen bond list. (Default value of *r\_maxHbondDistance* parameter is 2.5 A.) The list is recalculated for each new loaded molecular object. Hydrogen bonds on display are colored according to their hydrogen–acceptor distances. The option only allows to display hydrogen bonds without corresponding molecular object. Longer and shorter H–X distances in the hydrogen bond are color–coded, from red to blue, respectively.

For ICM object the hydrogen bonds are calculated much faster.

See also: undisplay hbonds, show hbonds.

## display label

```
display[{ atom | residue }] label [selection]
```

```
display variable label v_selection
```

a graphics label with atom name, residue name, variable name for all or selected atoms, residues or variables respectively. The text of this label is not user-defined, although you can control it in two different ways. First, residue label style can be set using either Ctrl-L in the graphics window or resLabelStyle preference, and variable label style either by Ctrl-V, or setting varLabelStyle

preference. Second, the ICM-shell string variable s\_labelHeader defines a prefix string for all labels. For example, if you display CPK atoms you may move the label to the right from the atom center by s\_labelHeader=" ".

The \_aliases file has convenient aliases (e.g. ds for display, unds for undisplay, re, for residue, va for variable) for those of us who like typing commands. In this case you may just type ds va la to display variable labels, etc.

Examples:

```
buildpep "FAHSGDH"
display residue label #
undisplay label
display residue label a_/his
display variable label v_//phi,psi
display variable label v_//* as_graph
display atom label a_/1:3/*
undisplay label
# or with aliases:
ds re la a_/1,3
unds la
.. etc.
```

## display map

```
display { map | map_name } [ I_colorTransferFunction ]
```

displays a real function defined on a three-dimensional grid (i.e., an electron density map). Optional iarray argument defines a color transfer function according to deviation from the mean. The color transfer function is explained in the color map section.

See also the color map command.

Example:

```
build string "se his arg"
make map potential "el" Box( a_/1,2/* , 3. )
display a_
display map m_el {3 2 0} # high values are invisible
make grob m_el 2.
display g_el
```

See also related commands: read, write, delete, list, show map, set, make (1), make (2) and file format icm.map.

## display movie : simulation trajectory

```
display movie [ sstructure ] [ image [ = s_framePath ] [ rgb | targa | png | gif ] ] [
s_MovieName] [ i_From [ i_To]] [ r_Smooth1 [ r_Smooth2]] [ as_1] [ center [ as_2 ] ]
```

lets you play, stop and reverse a Monte Carlo simulation trajectory as well as write a series of images for future assembly of those images into movies. To obtain the movie info use

#### read movie s\_MovieName

Integers *i\_From* and *i\_To* specify the frame range. Real values *r\_Smooth1* and *r\_Smooth2* determine minimum and maximum smoothing parameters (i.e. number of additional frames, inserted if conformation change is too dramatic). Specifying atom selection  $as_1$  defines a certain fragment on to the initial conformation, of which subsequent conformations are superimposed. Option center with selection  $as_2$  determines a fragment for graphics window centering (all, if center without  $as_2$ ). When playing a movie, you can use ICM interrupt (Ctrl-\) to stop, and then toggle stepwise frame playing, reverse, or quit playing. The default is to play a whole movie without smoothing, superimposition or centering.

Option image allows to automatically save a series of image files in the specified directory s\_framePath or in the default s\_tempDir directory. Allowed image formats are: rgb, targa, png, gif. The file extensions will correspond to the image file format. The image file names consist of the default path and name, appended with the frame number. Example:

```
display movie image="/tmp/f"
/tmp/f_1.png
/tmp/f_2.png
...
s_tempDir = "/home/jack/X"
display movie image rgb
/home/jack/X_1.rgb
/home/jack/X_2.rgb
...
```

All the other image preferences may be predefined by the IMAGE table.

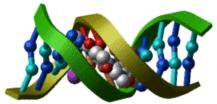
Option sstructure will dynamically reassign secondary structure while going through conformations of each frames. This option is very useful if you perform peptide/protein simulation and want to see if secondary structure elements are forming transiently.

```
See also: movie file.
```

Examples:

## display ribbon

display ribbon [base] rs\_color



display ribbon. Option base is used for displaying cartoon representations of the bases on the DNA/RNA ribbons.

See also base, GRAPHICS.dnaBallRadius, GRAPHICS.dnaStickRadius, and other dna settings in GRAPHICS.

## display site

display site rs\_color

display site information. Switch between different types of the site information with the SITE.labelStyle preference.

## display skin or dotted surface

```
display { skin | surface } as_1 as_2
```

display analytical molecular surface, also referred to as skin, or solvent accessible surfacearea. Each display skin command will delete the previously displayed skin in the current plane. To display several different skins, use the set plane command to change the current graphics plane before you issue the display skin command. You can also convert the skin into a grob with the make grob skin command. You can co-display many grobs on the same plane, as well as make the grob transparent. This grob can be further split into individual shells with the split command. See also: How to display and characterize protein cavities.

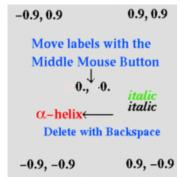
Example:

```
build string "se ala his glu" # test tripeptide
display # the wire model
display skin a_/1 a_/1 # skin around the 1st residue or just press <F1>
set plane 2 # key with your cursor in the graphics window
display skin a_/3 a_/3 # skin around the 3st residue
# now you can toggle planes with F2 and F1
# solvent-accessible surface
```

## display string

```
display s_StringText [ auxiliary ] [ color font= i_FontSize ] [ r_XscreenPosition
r_YscreenPosition ]
```

display a text string	in the graphics	window. Re	elative X and Y	screen
and pray a come burning	m me Brapmes			



coordinates (ranging from -1. to 1.) of the string beginning may be specified to display the string in a given location. Defaults are x = -0.9, y = 0.9, i.e. upper left corner of the screen.

The string can be dragged later to any location by the middle mouse button. Two fonts are at your disposal: the default font (usually times) and the auxiliary font (usually symbol). Both fonts can be redefined by the set font command. You can also switch to the auxiliary font and back inside the string by backslash-A (A). (.e.g "Red: AaA-helix"). You can also list and delete your string labels by the list label and delete label commands.

Examples:

display "Crambin"	# a simple string
display "Act.site of \Ab\A-lactamase" yellow	# Greek beta letter
display Name(a_1.) red 28, 0. 0.9	<pre># first object name # in the middle # (font size=28)</pre>

## display tethers

display tethers [ as\_ ] [ r\_minDeviation ]

displays tethers assigned to the selected atoms *as*\_ with deviation larger than *r\_minDeviation*. Tethers can be imposed between atoms of an ICM–object and atoms belonging to another object, which is static and may be a non–ICM–object. (0. by default).

## display window

display window [ i\_xLeft i\_yDown i\_xSize i\_ySize ]

displays an empty window of the specified size and position. This command is convenient for demo scripts. See also: set window

# 2.20.19. elseif

elseif

is one of the ICM flow control statements, used to realize conditional statements. See also: if, then, and endif.

## 2.20.20. endfor

endfor

is one of the ICM flow control statements, used to perform a loop in ICM-shell calculations. See also for .

# 2.20.21. endif

endif

is one of the ICM flow control statements, used to realize conditional statements. See also if, elseif, and then .

# 2.20.22. endmacro

A command ending a macro.

Examples:

# 2.20.23. edit

edit icmShellVariable

interactively edit the ICM-shell variable using your favorite editor defined by the s\_editor variable.

Examples:

edit mncalls # actually it is easier to type: mncalls=333
edit FILTER # edit a system table, do not change names of components
#
read table "pdb1" # read table with search results
edit SR # edit table SR

# 2.20.24. endwhile

endwhile

is one of the ICM flow control statements, used to perform a loop in ICM-shell calculations. See also while.

# 2.20.25. exit

exit[s\_message]

exit from a script file to interactive mode. Do not confuse this command with the exit option in, say, highEnergyAction preference.

Similar to return [ error s\_message ] from a macro.

To quit the program, use the quit command.

## 2.20.26. find

a family of commands for sequence and pattern searches, chemical matching, 3D pharmacophore matching, and alignment optimization.

## find alignment : automated structural alignment

```
find ali_initial [ superimpose ] [ r_threshold= 3. [ r_retainRatio= 0.5] ]
```

find the best structural alignment of two proteins by refining the inaccurate initial alignment *ali\_initial* with the goal of finding the largest possible subset of residues which have similar local backbone fold in 3D space.

Option superimpose automatically superimposes molecules according to the found structural alignment upon completion of the iterations. This command needs a starting alignment of 2 sequences linked to the molecules with at least one atom per residue. If Ca atoms are not found the atoms carrying the residue label (see the set label command) are used.

Low gap penalties of 1.8 and 0.1 are recommended for the initial sequence alignment.

**Algorithm :** At each step aligned pairs of atoms which are further than  $r_threshold$  from each other are disconnected so that at least  $r_retainRatio$  pairs are be retained. Then the molecules are superimposed again and new residue pairs are tested and accepted if it leads to a lower overall rmsd. Warning: the result strongly depends on the relevance of the starting alignment to the best 3D alignment. Sometimes 3D irrelevant sequence alignment pairs do not tend to disconnect to allow transformation into a global 3D alignment: e.g. if only one pair of elongated helixes is aligned in the starting alignment and it is only a small part of an optimal alignment which would be completely different, it might not be eventually found.

See also other types of structural searches and superpositions:

- find segment, fold search finds structural similarity on the basis of secondary structure elements (no sequence).
- qsearch and find pdb: search a database of a single structure for a fragment with a given sequence pattern and partial structural similarity (e.g. loop ends match).

• superimpose: performs structural superposition, the command can do it on the basis of sequence alignment on the fly.

Example:

```
read pdb "lnfp"
read pdb "lbrl.b/"
rm !Mol(a_*./A)
make sequences a_*.
aa=Align(lbrl_1_b lnfp_ml)
ds a_1.//ca,c,n grey
ds a_2.//ca,c,n green
superimpose a_1.1 a_2.1 aa
center
find aa superimpose
show aa
gapExtension=0.05
ab=Align(lbrl_1_b lnfp_ml)
find ab 4. 0.7 superimpose
show ab # better
```

## find database: sequence and pattern searches

find database [ unique ] [ delete ] [ write ] s\_databasePath [ exact [ margin= i\_nOfMutations]][ seq\_1 .. ] [ ali\_1 .. ] [ output= s\_projectName ] [ r\_probabilityThreshold ]

find database [unique] s\_databasePath pattern={ s\_patterns }[ output= s\_projectName] [ r\_probabilityThreshold ] [ margin= i\_seqMargin ]

fast sequence or pattern search through a sequence database. The sequence search program performs a full gapped optimal sequence alignment, which is a global alignment with zero-end-gap penalties (ZEGA). These alignments are more rigorous (not heuristic) than popular BLAST of FASTA searches. The latest statistics of structural significance of sequence alignments derived for a number of residue substitution matrices will be applied (Abagyan and Batalov, 1997) to assess the probability that a matching fragment shares the same 3D fold. The pattern search identifies sequence patterns (e.g. "A?[LIV]?\{3,5\}[!P]") in database sequences.

Arguments:

- exact performs a very fast search for identical or almost identical sequences. The margin= *i\_nOfMutations* parameter specifies the allowed number of mutations.
- option unique makes the program ignore hits with sequences 100% identical to the query set (if one sequence is a fragment of another, they are is still considered 100% identical).
- option delete will overwrite the output files without asking, as if 1\_confirm=no.
- option write is used to export ALL sequences from the blast-formatted files into to an external FASTA file defined by output= string (default *s\_databasePath.seq*). This option is the inverse of the write index sequence command which creates several BLAST files from a FASTA file.
- *seq\_1* .. (list of sequences) or *ali\_1* .. (list of alignments) determines which sequence will be searched against the database. The default (no argument) means all the sequences currently present in the ICM-shell (see list sequence).

- *s\_projectName* defines the names of the output files:
  - ◆ *s\_projectName.tab* a sorted table of hits (see below)
  - ◆ *s\_projectName.seq* a sorted list of sequence truncated to the matching fragment.
- *s\_databasePath* (default: 'swiss' files in the \$BLASTDB directory) defines the path of the three files with the compressed sequence files. For compatibility these three files (.bsq, .atb, .ahd) are the same as generated by the setdb (BLAST) command. The available files can be vied with the list database command, by default the "swiss" file is taken from the \$BLASTDB directory. If the environment variable \$BLASTDB is set, the three files will be taken from this directory. To read database files from any directory, specify its explicit path (e.g. "./myLocalDb" or "/home/user/myHomeDb1")
- $r_probabilityThreshold$  (default 0.0001) defines the lowest acceptable probability of hit. Threshold of 10/DatabaseSize is usually a safe threshold (no guarantees though). Practically  $10^{-5}$  is a safe threshold for a SWISSPROT search (65,000 sequences). At  $10^{-4}$  you may find interesting hits, but a more serious analysis may be required to confirm its significance.
- option margin= *i\_seqMargin* in the pattern search defines the length of flanking sequences added to the matching fragment and saved in the *s\_projectName.seq* file for further retrieval. Specify a very large number to store complete sequences.

Other important variables:

- alignMinCoverage (default 0.5) a threshold for the ratio of the aligned residues to the shorter sequence length.
- alignOldStatWeight (default 1.) a parameter influencing the statistical evaluation of sequence comparison. To use run-time statistics use alignOldStatWeight=0.
- Up to mnSolutions hits will be retained in the final table of hits.
- The parallel version of the program will use nProc CPUs (but not more than is available in your computer). The expected time is inversely proportional to the number of CPUs.
- maxMemory is a real ICM-shell-variable defining the size of the database buffer memory in Mb used by the command. If this size is smaller than the database, the sequences will be loaded in chunks.

The output table looks like this and contains the following fields:

#>T SR									
#> NA1 NA2	MI	MX	LMIN	LN	н	ID	SC	pP	DE
1hiv_a POL_HV1H2	57	155	99	0.665	0.099	100.0	103.69	30.00	"POL PROT"
1hiv_a POL_HV1BR	69	167	99	0.664	0.098	99.0	103.62	30.00	"POL PROT
<i> lines skipped</i>	</td <td>i&gt;</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>	i>							
1hiv_a POL_MLVAV	9	102	99	0.648	0.078	27.3	23.41	5.31	"PROTEASE"
1hiv_a VPRT_MPMV	172	272	99	0.799	0.290	29.3	21.95	4.82	"PROTEASE"
1hiv_a VPRT_SRV1	172	269	99	0.799	0.290	27.3	21.65	4.72	"PROTEASE"
lhiv_a GPDA_RABIT	33	145	99	0.785	0.264	28.3	20.98	4.50	"GLYCEROL3P"

- NA1 the query sequence (a single command can search several query sequences)
- NA2 the name of the database sequence
- MI : MX the matching fragment boundaries in the database sequence
- QMI : QMX the matching fragment boundaries in the query sequence
- LMIN the shortest sequence length in a pair (query, database sequence)
- LN log–correction factor (not used in pP but you may want to use it to resort the table).
- $\mathbf{H}$  the fraction of the database sequence covered by the alignment with the query. If you search against a database of domains this number should be close to 1 (e.g. the hit is less significant if

your query is only a part of a domain). It can be taken into account by multiplying pP by this number.

- **ID** percent sequence identity (number of identical residue pairs in the alignment divided by LMIN)
- SC normalized alignment score which is used to calculated the Probability. The score depends on the residue substitution matrix and gap penalties. (see the Score function).
- **pP** = -log<sub>10</sub> (Probability)
- **DE** the database sequence definition

#### Examples:

```
read sequence "GTPA_HUMAN.seq"
find database "/blast/swiss" output="gtpa1"
find database pattern="C?[DN]?\{4\}[FY]?C?C" "/blast/brku" margin = 5
unknown1=Sequence("TTCCPSIVARSNFNVCRLPGTPEAICATYTGCIIIPGATCPGDYAN")
find database exact unknown1 "/blast/brku" margin=1
```

### find molecule: chemical substructure search

```
find molecule s_Smile1 { s_Smile2 | S_Smiles2 }
```

Identify a complete match of the source molecule represented by a smiles string in another smiles string or an array of smiles strings representing a database of chemicals. Make sure that you unselect hydrogens in your smiles string. The following setup is optional:

- prepare the target strings with the Smiles ( a\_//![hdt]\* ) function (exclude hydrogen, deuterium and tritium)
- search the source string made without hydrogens

Only up to mnSolutions hits will be retained in the final table of hits. Change this shell variable if necessary. The function will return the results in the following variables:

- i\_out contains the number of hits
- I\_out contains the integer array of the hit numbers

An example:

```
squery = "CC=1C(=0)C=CC(C=1)=O"
read sarray s_icmhome+"chemDbSmiles"
find molecule squery chemDbSmiles # check I_out and i_out
```

The smiles string can also be generated with the Smiles function after conversion, e.g.:

```
read mol s_icmhome+ "ex_mol"
convert a_1.
squery = Smiles( a_1.//!h*,d*,t* )
```

find molecule [ tether ] ms\_1 [ ms\_2 ]

You can also use the alternative set of arguments and use molecular selections instead of the smiles strings. The atoms aligned to each sequential atom of the query molecule will be stored in the S\_out array.

**tether option:** After the equivalent sets of atoms in two molecules are identified, tethers can imposed pulling equivalent atoms of *ms\_1* to ~ms\_2. Also, if option tether is specified, the matched query atoms will be tethered to the target atoms, so that atoms can be later superimposed

An example:

```
build string name="a" "se nter his cooh"
build string name="b" "se nter ala his trp glu cooh"
find molecule a_a./his/!h* a_b. tether
display a_*.
minimize "tz" a_//?vt*
```

See also: Smiles function and the build smiles command.

### find pdb: fragment search

find pdb rs\_fragment os\_objectWhereToSearch s\_3D\_align\_mask [ s\_sequencePattern [
s\_SecStructPattern ] ] [ r\_RMSD\_tolerance]

A PDB database searching engine. Find a fragment (e.g. a loop) with certain geometry, sequence and/or secondary structure. Arguments:

- *rs\_fragment:* the search fragment template
- *os\_objectWhereToSearch:* the other object. In the qsearch macro this argument contains the current object from the qsearch database (see also s\_qsearchDir directory containing converted pdb-objects).
- *s\_3D\_align\_mask:* marks the residues to be used in the 3D superposition and comparison in terms of *r\_RMSD\_tolerance* (see below). The number of '**x**'s (or 'ON' bits) in the mask must be equal to the query fragment length (it may be discontinuous), while the total mask length should be equal to the found fragment length. For example, if you search for an 11–residue loop with the same geometry of 3–residue ends, but any geometry of the middle part your mask must be "xxx----xxx". If you want to match geometry of the middle part you would invert the mask: "---xxxxx---", etc.
- *s\_sequencePattern:* Use " \* " for any sequence. Otherwise you may use regular expressions, for example: "?A[!P]???\$".
- *s\_SecStructPattern:* Use " \* " for any secondary structure pattern. Otherwise, specify a regular expression, for example "?HHH\_\_\_EE[!\_]".
- *r\_RMSD\_tolerance:* RMSD threshold to accept a fragment as a solution. To avoid time-consuming optimal 3D superposition during the search, **distance Rmsd** (i.e. root-mean-square deviation between two Ca-atom distance matrices of the compared fragments) is used as a measure of spatial similarity on the preliminary stage of each comparison. However, in the resulting list of hits, collected in *SearchSummary* string array the optimal 3D

superposition coordinate Rmsd is presented. Therefore, RMSDs in the output list may exceed the specified threshold.

Hits will be stored in s\_out. The following just illustrate the syntax, it does not make much sense, since you need to loop through a database of objects to find something interesting.

Example:

```
read object "crn"
read object "complex"  # object in which to search
find pdb a_1./16:18,20:22 a_2. "xxx----xxx" "C?[LIVM]????G??" "*" 2.5
```

See also: qsearch macro.

#### find prosite or profile

```
find prosite [ append ] seq_[ r_minScore] [ i_mnHits]
```

find matching prosite patterns, store results in the SITES table. Option append indicates that the results should be appended to the existing SITES table. The default  $r\_minScore$  is 0.7. The default  $i\_mnHits$  is defined by the mnSolutions parameter.

Examples:

```
read sequence "zincFing.seq"
find prosite lznf_m
show SITES
```

## find pattern

```
find pattern [ number ] [ mute ] s_sequencePattern [ i_mnHits] [ { os_objectWhereToSearch |
seq_Name | s_seqNamePattern } ... ]
```

find specified sequence pattern (i.e. "[AG]????GK[ST]" for ATP/GTP-binding site motif A) in sequences or objects. Hits will be stored in s\_out . r\_out contains the number of found hits divided by the expected number of hits, as suggested by random distribution of amino-acids with frequencies from the Swissprot database. This "found/expected ratio" is also reported if l\_info=yes. If this number is 1. it does not mean anything, 10. means that you can publish the finding and two paragraphs of speculations, 10000. means that somebody else has already found this hit. Pattern language:

- ^ sequence beginning
- \$ sequence ending
- ? one character
- \* any number of any characters
- [ACD] alternatives
- [!ACD] all but the specified residues
- *char*  $\{ i_{min}, i_{max} \}$  : repetition. E.g. ? $\{5,8\}$  from 5 to 8 of any character.

Other arguments and options:

- number just report the number of hits instead of reporting each match
- mute suppress terminal output (used in scripts)
- *i\_mnHits* (default mnSolutions)
- *os\_objectWhereToSearch* the target molecular object.
- *seq\_Name* the target sequence. By default, the search is performed among **all** currently loaded sequences.
- *seqNamePattern* the target sequence name pattern to search through many sequences loaded to the shell.

Returned values

- s\_out text output of all matches
- i\_out the number of hits
- $r_{out}$  the ratio to the random expectation (it  $r_{out}>1$ ). it means that the number of hits is larger than the random expectation).

See also the searchPatternPdb macro.

Examples:

See also: read prosite, s\_prositeDat.

#### find segment: database search for a similar protein fold

```
find segment ms_source [ os_objectWhereToSearch_2] [ r_maxRMSD ] [ i_minNofAlignedResidues [
i_accuracyParam ] ]
```

find similar 3D structural motif with the given minimal length. The topology of the specified molecule *ms\_source* is searched for either in the second object or, by default, in the whole segment databank (foldbank.seg) which currently contains 2177 protein folds. Only structures with unique polypeptide sequences are included in the fold databank. You may create, save or append to the foldbank.seg a simplified topology description of your own file or a set of files (see write segment, read segment). The optional *r\_maxRMSD* argument specifies a root-mean-square deviation threshold of the coordinates of the ends of secondary structure vectors (default 3.0 A). Default *i\_minNofAlignedResidues* is 30 residues. Argument *i\_accuracyParam* (do not use it without special needs) defines the number of distances for each reference point which are taken into account in the distance RMSD calculation. Default value equals 3 (distances).

Examples:

## 2.20.27. fix

fix vs\_

fix (exclude from the free variable list) specified variables (such as bond lengths, angles and phases or torsions) in an ICM-object. This operation can be applied to the current object only (use set object os\_object first). See also: unfix.

Examples:

set v\_//omg 180. # set all omega torsions to the ideal value fix v\_//omg # fix all omega torsions fix v\_/8:16,32:40/phi,PSI,omg\* # fix the backbone in two fragments

Note using PSI torsion reference for correct residue attribution.

## 2.20.28. for

for

is one of the ICM flow control statements, used to start a loop in the ICM-shell. See also while, endfor .

## 2.20.29. fork

a powerful tool for parallelization of ICM-shell scripts.

```
fork [ i_nExtraProcesses ]
```

spawns one or the specified number of extra copies of ICM. This command will only work in a non interactive mode, i.e. you should run icm like this:

icm \_multiProc # from the unix shell or unix icm \_multiProc # from the interactive ICM-shell

The iProc shell variable will contain the current process number. The parent process has iProc = 0.

An example script \_multiProc with a hypothetical macro bigDatabaseJob which takes two arguments: the number of database chunks, the current chunk number, and the output file name:

```
read libraries
macro bigDatabaseJob i_nChunks i_Chunk s_outFile # definition
...
endmacro
read sequence "hot"
fork 4
```

# 2.20.30. fprintf

fprintf [ append ] s\_file s\_formatString arg1 arg1 arg2 arg3 ...

formatted print to a file. The specifications for *s\_formatString* are described in the printf command section. In contrast to the print and printf commands, the result of the fprintf command is not shown. E.g.

fprintf "a.txt" "%s\n" "Day Temparature" fprintf append "a.txt" "%s %.2f\n" "Monday", 22.4 fprintf append "a.txt" "%s %.2f\n" "Tuesday", 27.334

# 2.20.31. goto

goto

is one of the ICM flow control statements, used to jump over a block of ICM-shell statements. See also break, continue.

## 2.20.32. group

## group sequence

```
group sequence [ { seq1 seq2 ... | s_seqNamePattern | alignment ] GroupName [ unique {
    i_MinNofMutations | r_MinDistance } [ delete ] ]
```

group sequences into a sequence group to perform a multiple alignment with the align command.

Option unique allows you to select only the different sequences. If no argument follows the word unique, only identical sequences will be dismissed, otherwise they will be compared and retained if the number of differences is greater than *i\_MinNofMutations* (integer argument) or the distance between two sequences is greater than *r\_MinDistance* (real argument). The comparison criterion is complex and has the following set of preferences which may be useful in extracting a representative subset of sequences from a PDB-database:

- longer sequence is better that shorter
- if the names contain the X-ray resolution 2 digit suffixes (like a19 and 9lyz24, for resolutions 1.9 and 2.4 respectively), higher resolution is better (1.9 is better than 2.4).

Note Resolution suffixes are added by the read pdb sequence resolution command

- higher number in a pdb-file name is preferable, i.e. 9lyz is better than 3lyz. (I would not die for this principle, though).
- identical chains are chosed alphabetically (e.g. 9lyz\_a is better than 9lyz\_b).

Suboption delete tells the program to delete from ICM-shell all the sequences which were found redundant by the unique option.

Examples:

```
read sequences s_icmhome+"seqs.seq" # load sequences
group sequence aaa
                                    # group ALL the sequences into aaa
group sequence seq3 seq1 seq2 aaa
                                    # explicit version of the previous line
align aaa
                                    # multiple alignment
read sequences s_icmhome+"azurins.msf" # some of sequences are very close
                                    # but not identical
group sequence myAzur unique 0.15
                                    # 0.15 is a Dayhoff-corrected minimum
                                    # intersequence distance threshold
group sequence myAzur unique 26
                                    # all sequence pairs differ in
                                    # more than 26 positions
group sequence myAzur unique delete # duplicates will be removed
```

### group sequence unique=: clustering, redundancy removal and assembly

group sequence unique= "nt,junk,simple,overlap[> nRes ]"] [ i\_wordLen=6 [ i\_dictDepth=10
[ i\_nofMutations=0 ]]] [ delete ] [ nosort ] [ { seq1 seq2 ... | s\_seqNamePattern | alignment ]
GroupName

If you read a very large **redundant** set of sequences and sequence fragments some of which may (i) overlap or (ii) be included in another sequence, you may want to remove all the redundant fragments, and merge the overlapping sequences into a smaller number of longer sequences. In a simple case, if the number of sequences is not too large (less than a few hundred), this removal of redundancies and fragments in your sequence set, can be performed with the group sequence unique ... command described in the previous section.

To work on much larger sequences sets and allows to merge overlapping sequences a more advanced algorithm is needed. This ultra-fast removal of redundant protein or DNA sequences, may also assemble the sequences into larger consensus sequences and is invoked by group sequence unique= "options" command.

The command returns the result as a sequence group *GroupName*. Will work on tens of thousands of sequences at once. The important features of the command:

- it can cluster/unique millions of sequences very quickly (your computer just needs enough memory).
- larger sequences incorporate the matching smaller ones.
- merged or absorbed sequence names are added to the description of their master unless nosort is specified
- merging (option "overlap") protein sequences requires sticky C-terminus letter 'X'
- the algorithm is based on a dictionary approach and allows to have mismatches
- matching rules:

- ♦ for proteins: any letter matches 'X', B=(D or N) and Z=(E or Q)
- ♦ for nucleic acids: any letter matches 'N'
- ♦ if possible, 'X','N','B','Z' are replaced by a more specific letter from the matched sequence

Options (they can be combined in a comma-separated string, e.g. "nt, simple, junk"):

- delete the non-unique sequences are deleted not only from the group but also from the shell
- nosort do not merge descriptions of the merged sequences
- unique= "simple" the fastest mode. It will eliminate only the exact duplicates.
- unique= "nt" means that DNA or RNA sequences are compared (the program assumes protein sequences by default and a corresponding *i\_wordLen* of six). This implies the alphabet of A,C,G,T (or U) and the word length should therefore be increased. The default nucleic acid sequence word length of 13 allows to fit the entire dictionary into the memory of 256 Mbyte. If your computer has less memory, reduce the *i\_wordLen* to a smaller value.
- unique= "junk" this option tells the program to remove sequences that do not contain any meaningful sequence. This means that they are mostly composed of 'X's or 'N's and the intermittent sequence is shorter than *i\_wordLen*. This option is almost always useful.
- unique="stripX" this option tells the program to strip X (or N for nucleotide) character stretches from the beginning and from the end of the sequence. Those will be compressed into just one character. Useful if your sequences were *dusted* or repeat-masked.
- unique="noX" this option tells the program to skip the sequence quality enhancements (replacement of 'X','N','B','Z' by a more specific letter from the other very similar sequence).
- unique="complement" with this option the complementary nucleic acid sequences will also be considered and removed if redundant. This option **can not be combined with the "overlap" option**.
- unique="overlap[>numberOfRes]" Merge overlapping fragments in in addition of deleting the subfragments from the set. The number of overlaping nucleotides or amino acids can be redefined, e.g. unique ="overlap>25"
  - Two aminoacid sequences are merged only if there is the overlap is greater than the threshold (12 aminoacids by default) and the overlapping C-terminal residue is 'X'.

An example of the allowed merge for protein sequences:

```
s1 VTIKIGGQLKEALLDXGADDTVLEEMSLPGX-----
s2 -----EALLDTGADDTVLZEMSLPGRWKPKMIG
result VTIKIGGQLKEALLDTGADDTVLEEMSLPGRWKPKMIG
```

If for some reason your ESTs do not terminate with 'X's, they can be added by the following procedure:

```
for i=1,Nof(sequence )
   sequence[i] = sequence[i] //Sequence("X")
endfor
```

• Two **nucleic acid sequences** are merged if the overlap is 30 by default. There are NO special requirements for an 'X' nucleotide flanking the sequences.

- *i\_wordLen* (6 by default, 14 if the unique="nt" option is specified). The length of a word in the dictionary. The memory occupied by the dictionary depends exponentially on his length.
- $i_dictDepth$  (10 by default) limits the number of sequence fragments referenced referenced from a single 'word'. This option prevents the dictionary from growing to much in memory (what the product of  $i_wordLen * i_dictDepth$ ).
- *i\_nofMutations* (zero by default) the maximal number of mutations/mismatches between sequences which are considered to be redundant.

See also:

- Trans( *seq\_* frame ) to translate a DNA sequence
- align new # to align a cluster and generate the consensus
- Find( *sequence*, *s\_keyword*) # to find a retired sequence with the *s\_keyword* in its title among the newly formed sequences
- show [color]  $ali_{\_}$

#### group table

group table [ copy | append ] [ t\_tableName ] [ array [ s\_name ] array [ s\_name ] .. ] header [ shellObj s\_name shellObj s\_name .. ]

create a new table from individual arrays or append new columns or table header elements to an existing table. This example shows how an ICM table including both header elements and columns may look like:

```
group table t {1 2} "a" {"one", "two"} "b" header "trash" "comment" 2001 "year"
 Info> table t (2 headers, 2 arrays) has been created
 Headers: t.comment t.year
 Arrays : t.a t.b
show t.
 #>s t.comment
                             # TWO HEADER ELEMENTS
 trash
 #>i t.year
 2001
 #>T t
 #>-a-----b------ # TABLE ITSELF
       one
two
   1
   2
              two
show t.comment t.year
 trash
  2001
```

Options:

- copy: make a copy of the original ICM-shell object and move it to the table.
- append: add specified ICM-shell objects to the table (default: overwrite).

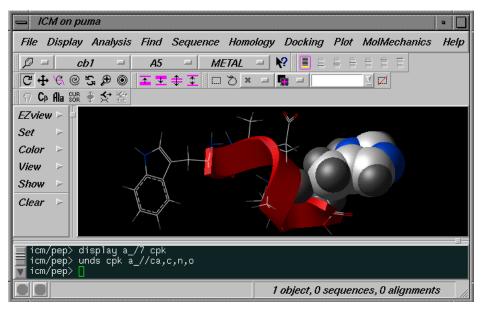
In the header section each ICM-shell object should be followed by a string specifying the variable name. The empty string will be interpreted as an indication to keep the name of the variable. Unnamed constants such as {1 2 3} or "adsfasdf" will be automatically assigned unique names. See also: split, Table.

More examples:

```
a=1
      # integer a
      # real b
b=2.
group table copy t header a "ii" b "rr"
      # create table t with t.ii and t.rr header objects
show t
group table t header a "" b ""
      # t with t.a and t.b header objects
show t
group table t {1 2 3} {2. 3. 4.}
      # t with automatically named table
      # arrays t.1 and t.2
show t
group table t {1 2 3} "a" {2. 3. 4.} "b"
      # t with table arrays t.a and t.b
show t
split t # split the table into individual arrays
```

# 2.20.33. gui

gui[simple]



start menu-driven graphical user interface from command line. The GUI runs the icm.gui file containing all the commands invoked by menus or pop-ups.

Option simple allows you to keep your terminal window separate from the graphics window.

% icmgl

gui simple

You can also invoke gui from the command line, e.g.:

icm -g # or icm -g mymenus.gui icm -G mymenus.gui # keep the original terminal window

#### **Terminal window and fonts**

icm -g (or gui command) invokes a GUI frame with its own built-in terminal window. To influence the font size in this terminal window, modify XTermFont record of the icm.cfg configuration file, e.g.

XTermFont \*-fixed-medium-\*-\*-\*-24-\*

If you prefer to keep the original terminal window use the icm -G option or invoke the gui simple command from ICM shell (I keep alias guis gui simple in my personal configuration file). In this case you can change the font of the terminal window with standard means of the window manager.

#### **3D** Graphics window

GUI has a GL-graphics window which can be undisplayed with the

undisplay window

command or from GUI by choosing Clear/No\_graphics menu item.

#### **Programming GUI**

To program GUI you need to modify a single file, called icm.gui. This file resides in the \$ICMHOME directory. The icm.gui file contains sections for each menu item. It the menu item requires a dialog, a widget is automatically generated.

The icm.gui section consists of the following fields:

Field	Example	Description
MENU	Display.Advanced.Pockets	this menu item will be added in the top menu bar
SIDE	Display.Advanced.Pockets	alternative to MENU, will be added in the side menu bar
OPTN	Apply	optional. Allows to specify special kinds of dialogs.
SYNT	display g_pocket	the ICM-command executed upon pressing 'OK' or 'Apply'.
An exan	nple:	

MENU Display.Color.White SYNT color white

To see how to specify arguments in a dialog, see the icm.gui file.

# 2.20.34. help

get help from icm.htm file. Set s\_helpEngine variable to "icm" (internal help in the text window), "netscape" or any other web-browser.

## help

help[input=s\_fileName][word1 word2 ...]

get full help in either text or html form, redirect it to the specified file, if the input option is specified. If you need help on a complex expression like read sequence, merge the words into one, e.g. help readsequence. Do **not** use plural forms of the nouns. Examples:

help Random help readsequence

The built-in help engine does not know about keywords. It is recommended to use the on-line version of the ICM manual which has a well-developed Index (download the newest version of the manual, man.tar.gz from the Molsoft ftp site).

### help commands

help commands [*s\_Pattern*]

generates concise list of syntax lines for all or specified commands.

#### help functions

help functions [s\_Pattern]

generates concise list of syntax lines for all or specified functions.

#### Examples:

help	<pre># type /stereo, and then letter n or Bar</pre>	
help help	# how to get help	
help commands	# list syntax of all commands	
help commands "rea*"	# list syntax of all read commands	
help functions	# list syntax of all functions	
help functions Matrix	# list syntax of the Matrix family of functions	
help help montecarlo help real constant	# start the browser to use its own search means # just the command name	
help readpdb		
help Split		

```
s_helpEngine = "netscape" # view help via netscape
help input="myHelp.htm" real constant
```

# 2.20.35. history

history[unique][i\_NumberOfLines]

display previous commands. Option unique squeezes out the repetitive commands. For example:

## 2.20.36. if

if

is one of the ICM flow control statements, used to perform conditional statements. See also: then, elseif, and endif .

## 2.20.37. keep

keep ICM-shell-variable-name1 ..

retain specified ICM\_shell variables or their classes (e.g. real, rarray etc.). This command is used in macros to avoid automatic deletion of all the local ICM-shell variables.

Also note that four classes of standard ICM-shell variables, reals, integers, logicals, and preferences, are automatically restored to their initial values by default. You can use the keep command to retain their new values.

Examples:

## 2.20.38. link internal variables of molecular object

link vs\_variablesToBeLinked

impose a chain of equality constraints (v[1] = v[2] = v[3] = ... = v[n]) on the specified variables (or, in other words, keep the specified variables equal to each other). If one of the variables is changed all the others will be changed. Energy derivatives are modified accordingly. This command is great for modeling periodic structures (e.g. (Pro-Glu)n).

Examples:

Be careful with selections of psi variables in peptides since they are assigned in ICM to the first atom of the *next* residue. PSI specification goes around that attribution.

## 2.20.39. link residues to sequences and alignments

link ms\_molecules { ali\_| seq1 seq2 ...}

link or associate protein molecules with separate sequences or sequences grouped in an *alignment*. If alignment *ali*\_is given, molecules are also linked to this alignment (note that the same sequence can be involved in several different alignments). Amino-acid sequences of *ms\_molecules* will be compared with specified ICM-shell sequences and identical pairs will be linked. Make sure that you specify **one** molecule selection, use logical or (|) between the two selections if necessary. Linking molecules with alignments allows an automatic residue-residue assignment by the following commands and functions: superimpose, set tether, Rmsd and Srmsd . Alignments can be prepared in advance either automatically by the align command or Align function, and/or modified by manual editing of the alignment file.

Use the ribbonColorStyle="reliability" option and color ribbon to display the local strength of the alignment. The strength parameter will be 3D averaged with the selectSphereRadius radius.

The following illustrates the first step of homology modeling.

Example:

```
build "newseq"  # that is what you want to build by homology
read pdb "template.pdb"  # that is the known pdb-template
read alignment "seq3Dali.ali" # prepared/modified sequence alignment
set object a_1.  # of the two structures
this is the first molecule that we
  # are going to model
link a_*. seq3Dali  # establish links between sequences
set tether a_1,2.1 seq3Dali  # impose tethers according to the alignment
minimize tether  # fold it according to the template
```

See also: 1\_autoLink

## 2.20.40. list

list[alignment][command][factor][function][grob][iarray][integer][ logical][macro][map][matrix][object][profile][rarray][sarray][ sequence][string][namel name2...]

list ICM-shell objects matching the name pattern (all if name-pattern is omitted). The plural form can be used for more natural expressions. 'list commands' actually means list all legal words known by ICM (ICM)

command words).

Examples:

```
list # list the "most wanted" object-types
list functions
list sequences # if you have aliases, you can
# type 'ls se' instead
list "*my*" # all ICM-shell variables containing "my"
```

# 2.20.41. list available sequence databases

list database

gives a list of BLAST databases which can be used by the find database command for fast sequence database searches. Normally, your system administrator should update the BLAST sequence files. ICM just needs a path to this directory which is defined by the \$BLASTDB system variable. The output of the command is saved in the S\_out array. This array can further be processed with the Field function.

Example:

```
list database
dblist = Field(S_out, 2) # sarray contains search databases
show dblist
a=Sequence("PDPPLELAVEVKQPEDRKPYLWIKWSP")
find database a dblist[2]
```

Trouble-shooting: If you get an error message, check the following:

- check if you have a directory with the blast-formatted files.
- make sure that your s\_dbDir variable is defined in your \_startup file and it contains the path to this directory (do not forget the last slash, e.g. /data/blast/dbf/). You can always assign it manually from the command line.

# 2.20.42. load

load things from the program memory (to load from disk files use read command). The opposite action to load is store.

## load conformation from stack

load conf i\_confNumber

assign the *i\_confNumber-th* conformation from the conformational stack and to the current object (e.g. when you browse conformations accumulated after a montecarlo run). If *i\_confNumber* is zero, the best energy conformation will be loaded. Montecarlo stack conformations are sorted according to energy values, however you may create your stack manually with an arbitrary order.

Note that the full energy of this conformation which had been stored in the stack can be accessed by the Energy ("func") function.

Example:

```
read stack "f1"  # read conformational stack
load conf 0  # set molecule into the best energy conformation
display a_//ca,c,n  # display the backbone
for i=1,Nof(conf)  # go through all the conformations
load conf i  # load them one by one
print Energy("func")  # extract its energy
pause  # wait for RETURN
endfor
```

## load movie frame conformation

load frame i\_movieFrameNumber [ s\_movieFileName ]

load specified frame from the movie. Note that the full energy of this conformation which had been stored by the simulation procedure can be accessed by the Energy ("func") function.

Examples:

```
build "alpha"  # build extended chain of the Baldwin peptide
read movie "alpha"
display movie "alpha" center # a-ha! conf in frame 541 is interesting
load frame 541 "fl"  # extract conformation from frame 541
print Energy("func")  # print its energy without recalculating
```

## load a structural alignment solution

load solution [ i\_solutionNumber ]

loads the specified solution previously stored by the align *rs\_residue1 rs\_residue2*.. command. The two output selections as\_out and as2\_out contain equivalent residues of the specified solution. The second object will be superimposed according to the Ca atoms of the found equivalent residues.

Example:

```
read pdb "4fxc"
read pdb "lubq"
display a_*.//ca,c,n
color molecule a_*.
align a_1.1 a_2.1 12 1.5 .1
center
load solution 2
                         # load the second best solution
color red as_out
color blue as_2^2 out
for i=1,10
  load solution i
  color molecule a *.
  color red as_out
  color blue as2_out
  pause
                         # rotate and hit 'return'
endfor
```

# 2.20.43. ICM-shell macros

### macro define a new macro command.

Syntax:

macro macroName [ mute ] prefix1\_macroArg1 [ ( default1 )] prefix2\_macroArg2 [ (default2 )] ...

# commands
# commands
endmacro

To invoke macro just type its name and provide arguments if necessary. Argument names should have explicit prefixes ( $i_, r_, s_, l_, p_, I_, R_, S_, M_, seq_, prf_, ali_, m_, g_, sf_, as_, rs_, ms_, os_, vs_, see above) to specify their type. If your argument list is incomplete, you will be prompted for the missing argument. Type q or enter$ **empty string**to quit the macro without execution. The following features make ICM macros extremely convenient to use:

- no need to explicitly define types of arguments (implicit definition by name)
- one may specify an arbitrary subset of arguments and in arbitrary order if the arguments have different types
- automatic prompting of the missing arguments
- an easy and flexible way to provide defaults in parenthesis after the argument
- automatic restoration of all the changed standard ICM-shell variables upon execution.
- new variables defined in the macro are local and will be automatically deleted upon execution, unless they are protected with the keep command.

Defaults can be provided in parentheses as simple constants (i.e.  $i\_window (8)$ ), or as the whole expressions (i.e.  $i\_1$  (mncalls)  $r\_a$  (Sin(\*2.))  $i\_2$ ). Default expressions can also be omitted.

Options

- auto automatically use defaults for the arguments missing in the command string. Example: nice "2ins". Since the second logical argument l\_wormStyle is missing its default value no will be used automatically.
- mute will suppress automatic prompting. Do not use parenthesized defaults with this option.

The predefined standard ICM-shell integer, real, and logical variables, as well as preferences (i.e. i\_out, l\_warn, wireStyle, PLOT.logo etc.) are be automatically restored upon completion, if changed in the macro, to retain the new value use the keep command. Note that the string variables should be restored explicitly. Many macros are supplied with the program.

Examples:

endfor endmacro

To invoke the macro, type

read object "crn" dsWorm a\_1 0.7

or just

dsWorm # and press Enter

A set of ICM macros is given in the \_macro file.

## 2.20.44. make

is a family of commands which create new objects of parts of them.

#### make bond: forming a covalent bond

make bond as\_singleAtom1 as\_singleAtom2 [ type= i\_type ]

adds a covalent bond between two selected atoms in a non-ICM molecular object (e.g. X-ray or NMR pdb-entries) or resets the bond type. The command is used to correct erroneous connectivity guessed by the read pdb command. This correction makes the molecule displayed in the graphics window look better and is necessary before conversion into an ICM molecular library entry (see icm.res or user library files) using the write library command. It can also be useful to display a connected Ca-trace. In interactive graphics mode you may type make bond and then click two atoms with the Control button pressed.

The type= option allows to set the bond type ( $i_type = \{1 | 2 | 3 | 4\}$ , for a single (default), double, triple and aromatic bond, respectively.

#### make bonds in an atomic chain

make bond *as\_chainOfAtoms* 

connects specified atoms in a linear chain. Useful for PDB entries containing only Ca atoms.

Examples:

See also: delete bond.

## make boundary: Poisson electrostatics

a command to prepare for the boundary element electrostatic calculation

make boundary [ as\_ ]

this is an auxiliary command which is required if you need to calculate the electrostatic free energy with the boundary element method several times. Optional atom selection  $as_{from}$  which the electrostatic field is calculated can be specified. This may be the case if the charge distribution changes but the shape does not. This command does not generate any output by itself, it just creates the internal table which can later be used by the show energy command or the Potential() function.

The dielectric boundary is a smooth analytical surface which is built with the contour-buildup algorithm ( toa96{ Totrov,Abagyan, 1996). The surface looks like the skin surface, but uses different radii which were optimized against experimental LogP data. Both skin and the dielectric boundary uses the same water radius ( the waterRadius parameter). The "electrostatic" radii used by ICM to calculate the boundary are stored in the icm.vwt file.

See also: REBEL, electroMethod, delete boundary, show energy", term "el", Potential( ).

Examples:

#### make disulfide bond

make disulfide bond [ only ] as\_atomSg1 as\_atomSg2

form breakable disulfide bonds between two sets of specified sulfur Sg atoms, regardless of the distance between them. Forming the bond means that two Hg hydrogens of Cys residues are dismissed, a covalent bond between two Sg is declared (but not enforced) and four local distance restraints (see icm.cnt) are imposed. These restraints are indeed local, since two Sg atoms only start feeling each other when they are really close, otherwise the energy contribution is close to zero. Option only causes deletion of previously formed disulfide bonds, otherwise the new one is added to the existing list of disulfide bonds.

Examples:

montecarlo

# MC search for plausible conformations

See also: delete disulfide bond and (important!) disulfide bond.

## make drestraint: extract distances structure

make drestraint as\_select1 as\_select2 r\_LowerBound r\_UpperBound r\_LowerCorrection
r\_UpperCorrection [ s\_fileNameRoot ]

create two files containing the list of all the atom pairs specified by two selections (i.e.  $a_* a_* - all$  the pairs;  $a_1//* a_2//*$  atom pairs between molecules 1 and 2 for which the interatomic distance lies between *r\_LowerBound* and *r\_UpperBound*.

Note: it is critical that both selections are in the **same** object. Only tethers can pull to atoms of a different object.

For each pair of atoms a distance restraint type is created with lower bound less than the actual interatomic distance by  $r\_LowerCorrection$  and upper bound greater than the actual interatomic distance by  $r\_UpperCorrection$ . This command can be used for example to impose loose distance constraints between two subunits.

The number of the formed drestraints is returned in the i\_out variable.

See also: set drestraint as\_1 as\_2 i\_Type

if you want to impose a specific drestraint.

Examples:

## make factor: FFT calculation of diffraction amplitudes and phases

make factor map\_Source { I\_3Maximal\_hkl | r\_resolution } [ s\_factorTableName [ s\_ReName [
 s\_ImName ] ] ]

calculate structure amplitudes and phases from the given electron density map by the Fast Fourier transformation. The table '*s\_factorTableName*' with h,k,l and structure factors will be created (further referred to as **T** for brevity). It will contain the following members:

- three integer arrays of Miller indices: T.h T.k T.l
- two rarray of real and imaginary parts of the calculated structure factors. Default names: **T.ac** and **T.bc**, respectively. Alternative names can be explicitly provided in the command line.

If structure factor table *s\_factorTableName* already exists, structure factor real and imaginary components are created or updated in place. Any other arrays containing experimental, derivative or control information

may be added to the table and participate in selections and sorting.

Example:

## make grob map command to contour electron density

make grob [ solid ] m\_map [ name= s\_grobName ] [ box ] [ I\_indexBox[1:6] ] [ { r\_sigmaThreshold |
exact r\_absThreshold } ]

Create graphics object by contouring electron density map at a given threshold.

**threshold:** By default the contouring level is calculated as the mean map value (returned by Mean ( *m\_map*)) plus mapSigmaLevel times root-mean-square deviation value. If a real value argument is provided, the mapSigmaLevel shell variable is redefined. Option exact allows to specify *absolute* value at the contouring is performed. Example:

```
buildpep "his glu"
make map potential Box( a_ 3.)
make grob m_atoms 3. # 3 sigmas above the mean
# make grob m_atoms .2 exact # countour at 0.2 level
# .2 or .1 exact is useful to detect almost closed pockets
display g_atoms
#
make grob m_atoms exact 0.15 # at value of 0.15
display g_atoms
```

Defaults:

- create simple chicken wire map (sections in three sets of planes, NOT solid)
- take the current map;
- generate the name of the grob which is the same as the map name except for the g\_ prefix;
- contour the whole map
- use threshold value from the ICM-shell real variable mapSigmaLevel.

Option solid tells the program to create a solid triangulated surface which can later be displayed by display grob solid command. The threshold is expressed in the units of standard deviations from the mean map value, i.e. 1.0 stands for one sigma over the mean. *I\_indexBox* [1:6] is optional 6-dimensional iarray containing { i\_startSection i\_startRow i\_startColumn i\_NofSections i\_NofRows i\_NofColumns }. It overrides the default, contouring the whole map.

Option box adds surrounding box to the grob.

## make grob image command to create a vectorized graphics object.

```
make grob image [ name = s_grobName ]
```

create a vectorized graphics object (grob) from the displayed wire or solid objects. The information about colors will be inherited. Very useful if you want to export wire, ribbon or CPK into another graphics program, since graphics objects can be written in portable Wavefront (.obj) format. Further, graphics objects can exist independently on the molecules which may be sometimes convenient. Also, underlying lines and vertexes can be revealed. The graphics object created from the displayed solid representations assigns and retains color information as lit in a given projection. These colors can not be changed. Use special make grob skin command to generate a more elaborate graphics object from skin.

Examples:

```
ds a__crn.//!h* ribbon # ribbon
make grob image name="g_rib"
display g_rib smooth only # try select g_rib and Ctrl-X,Ctrl-E/W etc.
# option smooth eliminates the jaggies.
write g_rib # save to a file
```

## make grob matrix

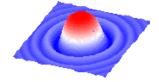
make grob [ solid ] [ bar [ box ] ] [ color ] M\_matrixName [ r\_istep r\_jstep] [ [ name= ]
s\_grobName ]

Create a three–dimensional plot from  $M_matrixName$ , so that x=i\*  $r_istep$ , y=j\*  $r_jstep$  and F(x,y)=  $M_matrixName[i,j]$ . Options:



- bar : generate rectangular bars for each i, j matrix value instead of a smooth surface.
- box : add a box around the 3D histogram
- color : color grob by value according to the PLOT.rainbowStyle preference.
- ٠

solid : tells the program to triangulate the surface



Examples:

```
read matrix s_icmhome+"def"
make grob g_def solid
display
# OR
read matrix "ram"  # phi-psi energy surface
make grob ram 1. 1. 0.1  # create the surface
display g_ram magenta  # display it
make grob solid ram 1. 1. 0.08 name="g" # create the surface
display g solid gold  # display it
```

#### make grob potential

```
make grob potential [ solid ] [ as_1 [ as_2] ] [ r_gridCellSize [ r_margin [ r_polentialLevel ] ] ] [ [
name= ] s_grobName ]
```

create graphics object of isopotential contours of electrostatic potential which takes not only the point charges but also the dielectric surface charges resulting from polarization of the solvent. This potential need to be calculated in advance by the boundary element algorithm. Contours can be displayed in the wire and solid representations (see also display grob). The default parameters are:

- r\_gridCellSize 0.5 A (you may want to increase it up to 2A for speed).
- *r\_margin* 5.0 A (you may want to reduce it for speed).
- *r\_polentialLevel* 0. kcal/mole/electron\_charge\_units.

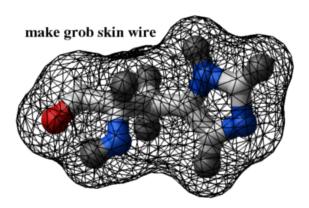
See also: make map potential, electroMethod, make boundary, show energy "el", term "el", Potential( ).

#### Examples:

```
build string "se his arg glu"
electroMethod="boundary element" # REBEL algorithm
make boundary
make grob potential solid 2. 4. 0.1 name="g_equipot1"
display g_equipot1 transparent blue
make grob potential solid 2. 4., -0.1 name="g_equipot2"
display g_equipot2 transparent red
ds xstick residue label
```

make grob skin [ wire ] [ as\_1 [ as\_2 ] ] [ [ name= ] s\_grobName]

create grob containing the specified molecular surface (referred to as skin). If the wire option is given the transparent wire grob will be created (solid grob is the default). It will have the same default color. The disconnected parts of this grob may later be split. The grob will be named by the default name g\_objName unless the name is explicitly specified.



A grob can later be colored with the color grob potential command.

Examples:

## make key

make key { s\_smiles | as\_ } [ S\_arrayOfFragmentSmiles ]

generates a binary chemical key, i.e. a bit-string in which each bit corresponds to a chemical substructure, converts the bit-mask into the hexadecimal string and saves this hex-string in s\_out. The bit-string with chemical substructure information can then be used to calculate the Tanimoto similary distance with another chemical key.

By default the make key command uses a built–in array of 96 substructures, and generates a 24–character hex–string ( each hex–character codes for 4 bits ), however any string array of subfragment smile–strings (  $S_arrayOfFragmentSmiles$  ) can be provided.

The hex-string can be converted back into an array of bits packed into integers with the  $Iarray( \{ s_chemkey | S_chemkey \} key )$  function.

The bit-distance between two arrays of bit-strings represented by two iarrays can be calculated with the Distance(Iarray( $S_1$  key) Iarray( $S_2$  key)  $i_nBits$  [key]) or Distance(Iarray( $S_1$  key) Iarray( $S_2$  key)  $i_nBits$  simple) functions, where the number of bits,  $i_nBits$ , is usually 96, unless you use a user defined array of fragments. There is also a weighted form of the chemical key distance (see the Distance function). By

default, or with the key option, the function returns matrix with the Tanimoto similary distance (0. all bits are the same, 1. no bits in common), while with the simple option the second chemical key is considered as a sub-fragment and the distance becomes 0. (identity) if the sub-frament is present in the first bit-mask.

Examples:

```
read mol s_icmhome+"ex_mol.mol"
build hydrogen
set type mmff
convert
smil = Smiles(a_)
make key s
skey = s_out
```

#### make map

make map R\_6cellParameters I\_3NofSteps [ R\_6box | I\_6box ] ["zxy"] [ as\_ ] [ name= s\_mapName ]

create an electron density distribution for atom selection as\_(all atoms of the current object by default) on a three-dimensional grid. See also make map potential for a rough electron density map. The electron density is calculated from the cartesian coordinates of the selected atoms using a 2-gaussian approximation. If the l\_xrUseHydrogen logical is set to no, hydrogen atoms are ignored. The following parameters are taken into account:

- the shape of the gaussian is influenced by the individual atomic b-factors (see set bfactor).
- addBfactor is added to individual atomic B-factors

*R\_6cellParameters* is a real array containing { *a b c alpha beta gamma* } parameters. Optional *R\_6box* or *I\_6box* arrays define the corner of the map box (closest to the origin) and its sizes ( { xI yI zI dx dy dz } or { *nx ny nz dnx dny dnz* }, respectively). The whole cell is taken by default.

Examples:

```
read object "crn"
make map {5. 5. 5. 90. 90. 90. 0.5 a_//ca,c,n
```

## make map factor : calculate electron density map from structure factors

make map factor [ T\_factor ] [ m\_map]

calculate an electron density distribution on a three-dimensional grid from a structure factor table of the Miller indices, reflection amplitudes and phases. Requires that the map is created before with the make map command. If optional arguments are not given the current map and/or current factor will be used. A new empty map can be created from an empty selection by the

make map a\_!\*

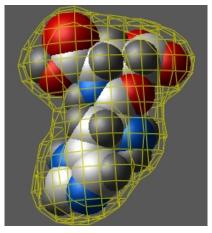
parameters # see the make map command.

# make map potential: grid energies

# make map potential [s\_terms][as\_][R\_6box][ r\_gridCellSize]

create a property map for the  $as_{-}$  selection. This command is used for low-resolution surface generation or to make grid potential maps for fast docking. The optional arguments are the following:

- *s\_terms* : a smooth gaussian atom density map is generated by default, otherwise the grid energy maps specified by the 2-letter terms are calculated, e.g. "gc,gh,gs,ge"). The names of the generated maps are standard and can not be changed.
- *as*\_ selection : All atoms of the current object are taken by default.
- *r\_gridCellSize* : by default is 0.5 A for small objects, the default increases with the size of the object. We do not recommend to use values over 7 A for very large objects.
- *R\_6box* : default it is a box around the selected atoms plus 3A margnins. The box defines coordinates of the two opposite corners of a box (see also the Box function).



m\_atoms contoured at 0.3 exact level. The 0.5 level is closer to the van der Waals surface.

• default (no terms specified): atomic density map m\_atoms; if contoured, m\_atoms generates a smooth gaussian envelope around a molecule (see Figures)

```
buildpep "his arg"
display cpk
make map potential Box( a_ 3.)
# wire surface
make grob m_atoms 0.3 exact # contours near vw-raduis.
display g_atoms
# solid surface
make grob m_atoms solid 0.5 exact
display g_atoms smooth
```

• term "el", map m\_el : Coulomb electrostatic grid, contributions truncated at +-100. kcal/mol.

```
build string "se his arg" "test"
make map potential "el" Box( a_/1,2/* , 3. )
display a_
display map m_el {1 2 3}
make grob m_el exact # contouring at 0. potential
display g_el
```

• term "gh" : van der Waals grid for a hydrogen probe, grid potential is truncated from above according to the GRID.maxVw parameter;

- term "gc", map m\_gc : van der Waals grid for a carbon probe; grid potential is truncated from above according to the GRID.maxVw parameter;
- term "ge", map m\_ge : electrostatic grid; grid potential is truncated from above and below according to the GRID.maxEl and GRID.minEl parameters;
- term "gb", map m\_gb : hydrogen bonding grid;
- term "sf", map m\_ga : surface accessibility grid. This map is not an independent term, but allows to correctly calculate atomic accessible areas if a part of the system is presented by the grid potentials. If a map named m\_ga is present it will be automatically taken into account in energy calculations of the "sf" term.

**Fine-tuning the maps** Sometimes you want the van der Waals grids, "gh" and "gc", generated from the whole receptor, while the "ge" or "gb" grids generated only from a small region of the receptor. In this case you can run the command two times with different source-atom selection.

Example:

```
make map potential "gh,gc" a_1 Box()
make map potential "gb,ge,gs" a_1/15:18,33:47 Box()
write m_ge m_gc m_ge # write three maps at once
```

An alternative method is to use the  $Bracket(m, R_{box})$  function which sets everything beyond the box to zero. Example:

make map potential "gh,gc,gb,ge,gs" a\_1 Box()
m\_ge = Bracket(m\_ge, Box( a\_1/15:18,33:47 )) # redefine m\_ge

#### make peptide bond

```
make peptide bond as_C as_N_or_S
```

form the peptide bond between two selected C- and N- atoms, or the thioester bond between C- and Satoms. The bonds may be formed between the terminal amino- and carboxy- groups (a\_/1/n and the last c), as well as between such amino acid side-chains groups as a\_/lys/nz and a\_/asp,asn/cd, a\_/glu,gln/cg. See also: delete peptide bond How to modify an ICM-object.

```
build string "se nh3+ gly gly gly gly his coo-"
display
make peptide bond a_/nh3*/n a_/his/c  # form a cyclic peptide
display drestraint
minimize "ss"
minimize "vw,14,hb,el,to,ss"
#
# form thioester bond
#
build string "se cys ala ala ala glu"
display
make peptide bond a_/1/sg a_/5/cd
```

minimize "ss" # term "ss" is responsible for the extra drestraints

#### make sequence: extract from pdb or icm structure

```
make sequence ms_[name= { s_name | S_names } ] [ resolution ]
```

creates sequences (or, more strictly speaking, ICM-shell objects of 'sequence' type) of residues composing selected molecules *ms\_*. One-letter equivalents of full residue names are specified in the icm.res library. Option resolution adds the X-ray resolution value multiplied by 10 to the name (e.g. 2ins\_a25 for resolution of 2.5A) or 'No' for NMR and theoretical structures. The resolution is not appended if option name= is specified.

The make sequence command also extracts both the secondary structure and the site information.

See also: read pdb sequence

Examples:

```
make sequence a_2ins.a,b  # two seqs 2ins_a and 2ins_b created
make sequence a_2ins.a,b resolution # resolution*10 added to the name
make sequence a_1.1 name="aa" # sequence named: aa
make sequence a_2ins.a,b name={"aa","bb"} # seqs named: aa and bb
```

#### make tree

make tree ali\_name [ s\_epsFile ]

reconstruct the evolutionary tree from the specified sequence alignment using the neighbor-joining method (Saitou and Nei, 1987). Create a PostScript image of this tree which will be saved in the *ali\_name*.eps file. See also: the align command.

Examples:

make tree M\_squareDistanceMatrix[1:n,1:n] [ S\_objectNames[1:n] ] [ s\_epsFile]

reconstruct the evolutionary tree for arbitrary objects from the **matrix of pairwise distances**. The names of individual objects may be provided in a string array for a nicer PostScript picture. This command is cool.

See also: Disgeo function.

```
read matrix "dist"  # read a distance matrix [n,n]
make tree dist
unix gs dist.eps
```

# make unique: reorder atoms in a unique order.

Example:

```
read mol s_icmhome+"ex_mol"
make unique
#
build hydrogen
set type mmff
convert
Smiles(a_) # unique smiles string
```

# 2.20.45. minimize

```
minimize[stack][i_mncalls][vs_][s_termString][as_1[as_2]]
```

minimize locally the sum of currently active, or specified, terms of the energy/penalty function with respect to variables specified by *vs\_*, or all the free variables, if variable selection is skipped.

# **Optional arguments:**

**stack :** If option stack is specified, the procedure extracts each stack conformation, minimizes it and replace the stack conformation with the optimized ones. The stack can be generated with the montecarlo procedure, manually created with the store conf command, or read from a stack file. This command allows to refine your set of alternative conformations all at once.

*i\_mncalls* : defines the maximal number of iterations. The minimization procedure can terminate earlier if the gradient becomes lower than the tolGrad parameter. If *i\_mncalls* is not provided, the default parameter mncalls defines the maximal number of function evaluations during the minimization.

 $vs_{-}$ : variable selection If selection of variables  $vs_{-}$  selection is specified, the object will be refixed but the initial fixation will be restored after minimization.

s\_termString : redefines the set of terms used in the minimization dynamically (e.g. minimize "tz"). You may check the active terms with the show terms command, or change them before the minimization with the set terms ".." command. By the way, the active terms can be shown as a part of your command line prompt if you add the %e specification to s\_icmPrompt variable (like s\_icmPrompt="icm/%o/%e> ").

**atom pair filter:** By default all the atoms and all the atom pairs within distance thresholds vwCutoff and hbCutoff are involved in the calculation. However, two explicit atom selections [ $as_1$  [ $as_2$  ]] may impose a mask on atom pairs involved in the calculation of the pairwise energy or penalty terms. The default for the skipped  $as_1$  is all the atoms. If only the  $as_1$  is specified, the  $as_2$  is assumed to be all atoms. Using atom selections is dangerous and is not recommended since there are many combinations which do not make sense and give unpredictable results.

the algorithm: the minimizeMethod preference. The type of algorithm (conjugate gradient, quasi Newton, or automatic switching between the two) is defined by the minimizeMethod preference. The progress bar will show you the progress of the procedure. If minimizeMethod="auto", the progress bar of the minimization procedure will show the 'C' character in a row of dots and colons when the quasi–Newton method switches to the conjugate gradient method.

Dots show progression of the minimization procedure, while colons mark recalculations of neighbor lists. The lists are updated if at least one of the atoms deviates from its previous position by more than 1.5 A. Both basic methods use the analytical derivatives of the terms with respect to free internal variables.

The procedure is terminated if the gradient falls below the tolGrad parameter or if the maximal number of function evaluations is reached.

the output l\_showMinSteps flag and i\_out: The actual number of function evaluations during minimization is saved in the i\_out variable. The l\_showMinSteps flag allows to see every iteration of the minimization procedure. To speed up the procedure you may switch off the l\_minRedraw flag to suppress redrawing of the molecule for each new conformation.

Examples:

buildpep "HHAS;TW"	<pre># create object from "def.se" sequence file</pre>
minimize v_//xi*	# do not touch the backbone torsions
minimize	# use all variables
minimize 500	<pre># run longer until number of calls is 500</pre>

# minimize cartesian: full conformational optimization

```
minimize cartesian [ stack ] [ type ] [ charge ] [ i_mncalls ] [ s_termString ]
```

minimize the mmff energy for a fully flexible molecule in the space of atomic cartesian coordinates. Before running this command please make sure that the atomic types and charges are set and the mmff libraries are loaded.

The *i\_mncalls* and *s\_termString* have the same meaning as in the previous command. Options:

- stack : if option stack is specified, the procedure extracts each stack conformation, minimizes it and stores back to the stack.
- type : if option type is specified the set type mmff command is executed and mmff atoms are assigned.
- charge : if option charge is specified the set charge mmff command is executed and mmff partial charges are assigned
- *i\_mncalls* : redefines the maximal number of minimization iterations (mncalls)
- *s\_termString* : allows to dynamically redefine the default energy terms.

Example:

```
build string "se nter his cooh"
display
minimize cartesian type charge
```

The drop and tolGrad minimization parameters will still apply.

# minimize loop after build model

minimize loop i\_loopNumber

to use this command you must run the build model command first. The build model command may not be able to find a perfectly matching loop. Two sorts of problems may appear: the imperfections of the loop attachments and the clashes of the loop to the body of the model.

The minimize loop command optimizes the covalent geometry at the junctions and the clashes through an interative procedure which maintains the loop closure.

The energy function used by the command is not as detailed as the full atom energy. It is advisable to perform a regularization (e.g. regul  $a_$ ) and full atom refinement.

To save all the graphical frames during this minimization set the autoSavePeriod variable to the special value of 99. In this case png image files named  $f_x_y$ .png, where x is the loop number and y is the frame number, will be saved in the current working directory.

#### minimize tether: threading a model with idealized geometry through a pdb-structure

```
minimize tether [ vs_ ]
```

regularization procedure. It creates a conformation (i.e. determines free variables) that minimizes distances between atoms and their tethering points. If initial model was built from standard amino-acids with idealized covalent geometry, this procedure will create a model with standard bonds and angles which fits the best to the target set of atom coordinates. The tethers may be imposed by the set tether command. An integer variable minTetherWindow defines the maximal number of preceding torsions which are locally minimized to best-fit the pdb-model. Optional variable selection *vs*\_ allows to perform fitting only for the selected fragment of the model. This may be convenient if you want to re-fit only a local fragment. Variable r\_out contains the RMS deviation between the template and the model.

# 2.20.46. menu

a tool for making clickable strings in the graphics window.

```
menu [ i_string1 i_string2 ... ]
```

this command declares the listed string labels as active and returns the chosen string number in i\_out. If no arguments are specified, only the last string will be "clickable". See also \_demo\_main file.

```
while(yes)
display string "Menu"
display string "Fish" -0.7, 0.6 yellow # 2
display string "Pork" -0.7, 0.5 yellow # 3
display string "Pasta" -0.7, 0.4 yellow # 4
display string "Quit" -0.7, 0.3 yellow # 5
menu 2 3 4 5
choice=i_out
delete label
```

```
if
      (choice == 2) then
  display "Good choice.\n Our fish is the best.\nClick here"
  menu
  delete label
elseif(choice == 3) then
  display "Good choice.\n Our pork is the best.\nClick here"
  menu
  delete label
elseif(choice == 4) then
  display "Good choice.\n Our pasta is the best.\nClick here"
  menu
  delete label
elseif(choice == 5) then
  quit
endif
endwhile
```

# 2.20.47. modify

modify chemical structure of a molecule by replacing one part with a specified group or "residue" from icm.res or user residue library.

# modify atom with a library group

modify as\_exitAtom s\_NewRadical

replace the branch starting from the specified atom by another library radical. Suitable for standard biochemical modifications, such as glycosylation, phosphorylation, etc. (Note that to myristoylate N-terminus you need to use "myr" as N-terminal residue, i.e. build string "se myr ala ala coo-").

Examples:

# modify: single or multiple residue mutations

modify rs\_ s\_NewResidueName

replace selected residue(s) rs\_ by another residue  $s_NewResidueName$ . The backbone conformation is not changed, unless the new residue is "pro" and the phi angle is outside [-90.,-30.] range.

You can replace amino acids (the usual list of three letter codes), as well as nucleotides: "rpa" "rpg" "rpc" "rpu" for RNA and "dpa" "dpg" "dpc" "dpt" for DNA.

Examples:

```
# Peptides and proteins
#
modify a_/15,18 "his"  # substitute residue 15 and 18 with histidines
modify a_/ala "val"  # substitute all alanines with valines
#
# DNA or RNA
#
read pdb "4tna"
convert
modify a_4tna1./66 "rpu"  # substitute nucleotide 66 by Uracyl
```

#### modify by grafting parts of objects

modify as\_atom1 as\_atom2

replace a fragment of the molecular tree starting from a specified single atom  $as\_atom1$  (e.g. a\_/15/cg) by a subtree starting from another single atom  $as\_atom2$ . This subtree is simply copied and not altered in any way. It is recommended to perform molecular building operation interactively and with your molecule displayed in the graphics window. Type modify and Ctrl-click the atom starting the branch to be replaced and then the atom starting the branch to be grafted. It does not matter where you take the modification group from. It may be the same molecule, a group in another object, etc. You may want to load a residue containing the group of interest directly from the icm.res residue library by doing.

Examples:

show residue types # find out what residues are available build string "se ret" # create a new object with retinol molecule.

After the modification you can remove objects (such as "ret" in the above example) used for construction. Be careful if modifying atoms within ring systems; the results may not always be obvious unless you know how the ICM-tree is constructed (you'll be kindly warned anyway). However, the whole ring can be modified or grafted without any difficulty.

Examples:

buildpep "MIPEAY" # build a molecule display # display it to click two atoms and watch modify a\_/1/ce a\_/1/ha # replace methyl group of Met-1 by a hydrogen modify a\_/2/hd13 a\_/2/cg2 # methylate hd13 hydrogen of Ile modify a\_/3/hg1 a\_/6/oh # turn proline into hydroxyproline

# 2.20.48. montecarlo

a generic command to sample conformational space of a molecule with

the ICM global optimization procedure.

montecarlo[OPTIONS][vs\_MC[vs\_minimize]][local rs\_loop
]

runs Monte Carlo simulation for specified variables *vs\_MC*, with local minimization with respect to the *vs\_minimize* variables following after each random move.

the gold

Where is

Each iteration of the procedure consists of

- 1. a random move of one of 4 types;
  - change one internal variable by a random value (e.g., montecarlo v\_//x\*)
  - change a group of angles described as a vrestraint according to its probability distribution (e.g. set vrestraint a\_/\*; montecarlo v\_//\*)
  - change the six positional variables (e.g. montecarlo v\_2//?vt\*) defining position of a molecule in space (the so called pseudo-Brownian move).
  - change the loop conformation (e.g.

```
set vrestraint a_{16:24} montecarlo v_{16:24} local a_{16:24}
```

- 2. local energy minimization;
- 3. calculation of the complete energy potentially including surface and advanced electrostatics terms ( REBEL or MIMEL);
- 4. acceptance or rejection of this iteration based on the energy and the temperature.

Three possibilities for variable selections arguments:

- no variable selections: both *vs\_MC* and *vs\_minimize* will be set to all free variables. Some vs\_MC variables, such as torsions rotating methyl groups, NH2 groups , will be automatically filtered out, since it is enough to just locally minimize them.
- one variable selection: the specified selection will be considered as the *vs\_MC*, *vs\_minimize* will be **the same** *vs\_MC*.



Pseudo-Brownian

• two variable selections: the first one is *vs\_MC* selection, the second one is *vs\_minimize*. **Important:** if two selections are explicitly specified, only *vs\_MC vs\_minimize* will be set free. It means that during the montecarlo procedure the object will be fixed differently than before. After the command, the status of variables will be returned as they were before the montecarlo procedure. There are two basic possibilities: unfix on the fly or unfix first and then run montecarlo:

```
montecarlo vs_MC vs_minimize # unfix on the fly
# OR
unfix only vs_minimize # prepare fixation (vs_MC is a subset of vs_minimize)
montecarlo vs_MC # now one selection suffices and
# the object set of free variables is not changed
```

#### **OPTIONS:** append

appends to the existing conformational stack (overwrites by default).

fast :

rapid side-chain optimization. This option allows to accelerate the calculation by minimizing only the strained variables after each step. Needs the selectMinGrad parameter to be set to 1.5. Example:

```
build string "se ala his trp glu" selectMinGrad=1.5 set vrestraint a_/* montecarlo fast v_//x*
```

This mode is useful for side chain optimization in homology modeling.

bfactor :

you can use the bfactor option to sample 'hot' parts of structure with higher probabilities. The relative frequences are taken from the b-factors of the atoms belonging to the mc-variables. Example:

```
buildpep "ala his trp glu" # default b-factor=20
set bfactor a_/2 1000. # make 2nd his hot
montecarlo bfactor
```

To preserve the old bfactors, save them before the simulation and restore after. E.g.

```
b_old = Bfactor(a_//*) # save
..
set bfactor a_/10:20 200.
montecarlo bfactor
..
set bfactor a_//* b_old # restore
```

local

local [ dash ] [ a\_/residueRange1,residueRange2... ]

( this option is specified after the main [ $vs\_MC$  [ $vs\_minimize$ ]]) option local makes local deformation type movement for specified regions (e.g. two loops a\_/15:22,41:55). Suboption dash chooses angles for random deformation symmetrically with respect to the loop center.

movie

records all accepted conformations sequentially in a binary \*.mov file. Later one can read movie, display movie, and operate with individual frames, e.g.

```
for i=1,Nof(frames)
    load conf i  # to extract a frame
    display skin white center
    write image png "f"+i
endfor
```

mute

suppresses the text output about every random move

 $r\_exitEnergy$  real argument determines if you want your procedure to exit upon achievement of equal or lower energy value . For example, if you know energy of the minimum, you may want to stop the search when this value is achieved. E.g.

buildpep "AHWEND" # hexapeptide set vrestraint  $a_{/*}$  # BPMC-probability zones montecarlo 10. # stop after energy of 10. is reached

#### two atom selections: montecarlo .. as\_1 as\_2

(option is NOT recommended for beginners) Atom selection arguments [*as\_select1* [*as\_select2*]] impose a filter on atom pairs considered in the terms of internal energy like "vw,el,hb,sf". There are three possibilities:

- no selections the whole object (all atoms) is considered (the default)
- *as\_select* interactions of the specified atoms with ALL atoms in the object.
- *as\_select1 as\_select2* interactions between two selections. For example, a\_dom1 a\_dom1 would consider only the internal energy of the domain dom1.

#### reverse

this option makes a more intellegent random move in singlechain or a multichain molecule. By default if an angle is randomly changed near the beginning of a molecule, the second part of this chain moves. With the reverse the random move can occur in such a way that a part of the chain *above* a randomly chosen angle will stay the same, while the chain *below* the angle will move. Actually, the parts will be compared by molecular mass and the heavier part will be more likely to stay where it is than the lighter part. The probability that a part stays static is proportional to the number of atoms of this part. It is important that the virtual variables ( $v_//?vt*$  are not fixed).

This option is very useful in docking, since the receptor is static and the moving molecule should try to preserve the majority of current interactions. Also, the reverse option helps if one simulates the N-terminus of a multichain protein, or a docking of a peptide to a protein. Example:

```
read pdb "laya" # read a complex
delete a_!1,2 # keep only SH2 domain and a peptide
convert # make an ICM object with hydrogen
set vrestraint a_/* # set prob. zones
montecarlo reverse v_2 v_2 # re-dock the peptide
```

If you move the **1st** molecule, do not forget to unfix the fvtl variables of all other molecules, e.g.:

```
unfix only v_1 | v_*//fvt1
montecarlo reverse
```

If you always want to keep the C-terminus static and move the N-terminus, use the superimpose option (see below).

superimpose as\_3atoms\_per\_molecule

superimposes new generated conformations after every move. Usually if you change backbone torsion at the N-terminus, the whole molecule moves. This option allows to generate conformational changes at the N-terminal part of a peptide while its C-terminus occupies the same position in space. After each random move the first 3 atoms selected in molecule(s) will be superimposed on their initial position and the 6 positional variables (v\_//?vt\*) will be updated accordingly. The setup:

- 1. unselect the virtual variables from the MC selection (v\_//!?vt\*)
- 2. specify three or more atoms beyond the N-term. of interest for superposition
- 3. add virtual variables to the minimization selection (it is usually the default) to allow positional adjustments during minimization (the movements of C-terminus are suppressed only in the MC move, not in the following minimization).
- 4. if minimization is used (mncalls > 1), make a copy of the molecule and tether the C-terminus to it.

#### Example:

```
mncalls = 1 # move N-term residues a_/1:5 and while keeping
            # the rest in the same position
montecarlo v_//!?vt* superimpose a_/6/c,ca,o
            # virtual variables should be available for minimization
montecarlo v_/1:3/!omg,?vt* superimpose a_/6/c,ca,o
# Now a more realistic example
build string "se ala his trp ala ala ala ala"
display
display residue label
mncalls = 200
copy a_1. "original"
set tether a_/5:7 a_original./5:7
set terms "tz"
set vrestraint a_/*
mncallsMC=100000
montecarlo v_/1:4/!omg,?vt* superimpose a_/5:7/ca
```

The following ICM-shell variables and commands are important for the procedure.

```
mncallsMC,mncalls,
```

- temperature,
- mcBell to make rs-zones narrower or wider than in icm.res file
- mcJump
- mcShake the average amplitude of the pseudo–Brownian move
- mcStep an amplitude of the unbiased step
- 1\_bpmc if no, makes simple random steps (one angle by a random value)
- 1\_writeStartObjMC if yes, write the starting object with its fixation and geometry to a file.
- mnvisits three limits and three actions follow
- visitsAction,
- mnhighEnergy,
- highEnergyAction,
- mnreject,
- rejectAction,
- vicinity,
- compare.

EXPLANATION OF THE OUTPUT (below are 3 example lines with numbered fields):

1 2	3	4	5	б	7	9	9	10	11	12	13	14
DY Visi	600	16	gln	xi3	70	-98	94	-322.04	-324.56	35	4.87	51559
	600	32	ile	BPMC	ipt	ipt	ipt	-324.56	-290.80	18	65.72	51577
_Y Visi	600	16	gln	BPMC	qmm	qmt	qmm	-324.56	-323.93	41	3.78	51618

1. DY = Down Yes, i.e. energy has decreased after change and new conf. is accepted

\_\_\_\_ = up no , i.e. energy has increased and new conf. is not accepted

 $_Y = up Yes$ , i.e. energy has increased, but new conf. is accepted

- 2. stack operation code;
- 3. current temperature in Kelvin;
- 4. number of selected residue
- 5. selected residue name
- 6. name of randomly selected angle or BPMC to indicate the biased probability move
- 7. internal coordinate value or name of the multidimensional zone before random change;
- 8. internal coordinate value or name of the multidimensional zone after the random change but before minimization;
- 9. internal coordinate value or name of the multidimensional zone after the minimization;
- 10. energy before the random change;
- 11. energy after the random change and subsequent minimization;
- 12. number of function calls made during minimization;
- 13. gradient RMS deviation (normal completion is with low or zero gradient);
- 14. total number of function calls in the simulation.

The logic of stack operations is the following. There are three possible events for each slot of a stack:

- 1. new slot creation
- 2. energy improvement of the current slot conformational family
- 3. replacement of the looser conformational family by a better energy conformation

The starting conformation is placed to the first slot, if the stack is empty. At every simulation iteration, distances (either coordinate RMSD or angular RMSD, as defined by the compare command) are calculated between the current conformation and all slots. If any of the distances is less than the vicinity parameter, then the energies are compared and if the current conformation has the better energy, the stack conformation is replaced by the current one, otherwise the visit counter of the slot is incremented. If no similar structures are found, the conformation is appended to the stack, i.e. a new slot is created. If the stack is full, i.e. number of slots reached mnconf parameter, then the worst-energy structure will be substituted by the current, provided the latter has lower energy. Otherwise, no action is taken and *number\_of\_high\_energy\_conformation* counter is incremented (see also mnhighEnergy).

# 2.20.49. move

# move ms\_molecule: change tree topology

#### move ms\_moleculeToReconnect as\_terminalAtom

changes the topology of the basic ICM-tree by reconnecting the first virtual bond of a specified molecule to a given atom. This allows you to move two molecules together as one rigid body. By default, all the molecules are connected to the origin [0,0,0] through virtual bonds. The molecule can be connected only to the terminal atom, usually a hydrogen. The molecule can not be connected to itself (naturally, do not even try it). This operation is defined only for ICM molecular objects.

Examples:

build "twomol"	<pre># two molecules connected by virtual bonds</pre>
	# to the origin
move a_2 a_1/1/hn2	<pre># graft the second molecule to a hydrogen</pre>
	# on another molecule
	<pre># now the second molecule will move together</pre>
	<pre># with the a_1/1/hn2 branch</pre>
	<pre># if you change v_1//?vt* variables</pre>

#### move os\_: merge two objects

#### move ms\_whatToMove os\_whereToMoveTo

move one, several or all selected molecules (*ms\_whatToMove*) to the specified object *os\_whereToMoveTo*. When all the molecules are moved from the source object, the empty object is deleted. The *ms\_whatToMove* molecule or object are **appended** to the end of the *os\_whereToMoveTo* object and their virtual torsion tvtl becomes virtual phase fvtl. This command is used to create one object from several components.

```
# 1st object
read object "crn"
# 2nd object
build string "se ala his leu"
move a_2. a_1.
# take the 2nd obj and merge
# it with the 1st one
read pdb "lsis"
```

# 2.20.50. pause

pause [ i\_numberOfSeconds ] [ s\_message]

suspends execution for specified number of seconds. If no argument is specified, the program will wait until RETURN is pressed.

Examples:

```
pause 5 "You have 5 secs" # pause for 5 seconds
display a_//ca,c,n # How to analyze the conformational stack
for i=1,Nof(stack) # for all stack conformation
load conf i # load and redisplay each of them
pause "Press Return. N"+i # gives you time to inspect the structure
endfor # go on to the next conformation
```

# **Debugging shell scripts**

The pause command also can set the program into a debugger mode in which you will be prompted to confirm each command by pressing RETURN. In the debugger mode the l\_commands flag will be automatically set to yes and restored upon quitting. This is how to do it:

- To start the debugger mode, add to your script: pause "START DEBUGGER"
- To quit the debugger mode, type or add to your script: pause "QUIT DEBUGGER"

# 2.20.51. plot

create a PostScript file with a plot.

plot { R\_Xdata R\_Ydata | M\_XmultpleYdata } [ S\_PointLabels ] [ S\_PlotAxisTitles ] [ { R\_4Tics | R\_8Tics } ] [ s\_epsFileName ] [ options]

- **Simple input:** Two compulsory arguments *R\_Xdata R\_Ydata* contain the X and Y coordinates. Both arrays may also be integer arrays.
- Matrix input: allows you to specify several data sets. The *M\_XmultpleYdata* matrix may contain either X, Y1, Y2,...Yn columns or just Y1, Y2,...Yn columns if option number is used. Matrix M[2,n] or M[n,2] is equivalent to the simple input *R\_Xdata R\_Ydata* (Note that function Histogram() returns such a matrix). Additional convenience: by default, different data sets will be shown in different colors and a panel with series/color correspondence will appear at the position specified by the PLOT.seriesLabels preference (choose "none" to suppress the panel). To avoid ambiguity do not use explicit *S\_PointLabels* with the matrix input.

• Axis and Tics: 8–array *R\_Tics*[1:8] contains information about X and Y axis: { *Xfrom, Xto, XmajorTics, XminorTics, Yfrom, Yto, YmajorTics, YminorTics* }. If only 4 numbers are provided, they are interpreted as { *Xfrom, Xto, XmajorTics, XminorTics* } while the Y axis tic marks are determined automatically. By default, if this argument is missing, the tic marks for both axes are calculated automatically. Example:

- **Title and legends**: string array *S\_PlotAxesTitles*[1:3+NofSeries] contains { "Title", "X title", "Y title" } in the simplest case. If multiple series are plotted using *M\_XmultpleYdata* or **number** *M\_multpleYdata* arguments, each series may be named with additional components of the array: { "Title", "X title", "Y title", "Y1 title", "Y2 title",...}.
- **Plot controls**: Optional *S\_PointLabels* has the same number of elements as *R\_Xdata* or *R\_Ydata* and may contain either string to be displayed at the corresponding X Y point, or control information about marker type, color and size. The control string must start with underscore (\_). To display both symbols and string labels, duplicate X and Y arrays (e.g. X//X, Y//Y) and supply the first S\_PointLabel section with the symbol information and the second one with the string label information.

Examples of string labels:

s={"lcrn", "2ins", "1gpu", "3kgb", "4fbr", "6cia"} # text labels: show as is s={"\_red SQUARE 0.4", "", "", "\_green DIAMOND", "", ""} # control labels s={"\_line" "" "" \_red line" "" "" "\_blue line" "" ""} # control labels

The empty string tells the program to inherit all the settings for the previous point. Individual components of the string label are (i) color, (ii) mark type and (iii) mark size. Omitted components are not changed. Allowed mark types: **line, cross, square, triangle, diamond, circle, star, dstar, bar, dot, SQUARE, TRIANGLE, DIAMOND, CIRCLE, STAR, DSTAR, BAR.** Uppercase words indicate filled marks.

#### **Options**.

- append append the plot to an existing plot file.
- display view the created postscript file with an external viewer defined by the s\_psViewer variable.
- grid, or grid="x", or grid="y" draw grid at the major tics for the specified axis. Default: for both axes ("xy").
- exact data points can reside exactly at a margin.
- regression draw linear regression line.
- frame draw NO frame around the plot (paradox isn't it? yeaah we are tricky).

- origin make origin at (0,0) point.
- link enforce 1:1 aspect ratio, equivalent to PLOT. Yratio = 1.0.
- comment = S xyXYtext this option allows to draw one line of text along all the four sides of the plot box. The string array may contain up to four strings  $\{s_x,s_y,s_x,s_y\}$ :
  - 1. s\_x: lower horizontal string, i.e. comment={"xxxxxxx"}

  - 2. s\_y: left vertical string, i.e. comment={"","yyy"}
    3. s\_X: upper horizontal string, i.e. comment={"",","XXXXXXX"}
    4. s\_Y: right vertical string, i.e. comment={"x","y","X","YYY"}

This option may be used to draw amino acid sequence around a contact plot box or a dot plot box.

• number generates the sequential numbering for X-array if this array is missing and sets a natural X tic style. In case of matrix input (see above) option number allows to omit the X-array.

String variable *s* epsFileName with extension .eps defines the name of a PostScript file where the resulting plot is to be written to. The default of *s\_epsFileName* is "def.eps".

Examples:

```
x = Rarray(90, 0., 360.)
                                   # an array of angles with 4 deg. steps
plot x Sin(x) display
plot x//x Sin(x)//Cos(x) display
                                   # quick and dirty way to have two data sets.
                                     # Now let us get rid of the defect
s = Sarray(2*Nof(x))
                                   # S_PointLabels for both arrays
s[Nof(x)+1] = "_red line"
                                   # restart line for the first point
                                   # of the second set
plot x//x Sin(x)//Cos(x) s display # much better
plot Transpose(x)//Transpose(Sin(x))//Transpose(Cos(x)) display
read object "crn"
crn_m = Sequence(a_/A)
                                   # a_/A ignores termini
plot comment=String(crn_m)+Sstructure(a_/A) number Turn(crn_m) display # try it
plot comment=String(crn_m)+Sstructure(a_/A) number Turn(crn_m) {"Turn prediction", "Res", "P"}
                                   # to see it again
unix gs def.eps
```

See also: Histogram, plotRama macro in the macro file, and examples in the demo plot file.

# 2.20.52, plot area: show matrix values with color

plot area M XY data options [S Title XY] [{ R 4Tics | R 8Tics }] [s epsFileName]

plot 2D data from the matrix and mark values by color. Other arguments are the same as in the plot command. Distribution of colors is controlled by the PLOT.rainbowStyle preference. By default the minimal and maximal values of matrix *M XYdata* are used as extremes for coloring. **Options**:

• color = R 2MinMax option allows you to enforce specific boundaries represented by the color range. For example, if you chose the "blue/red" PLOT.rainbowStyle the matrix value smaller than or equal to the first element of the *R* MinMax array will be colored blue, while the matrix values larger than or equal to the second element of the array will be colored red, the middle values will be color with intermediate colors. The real array of boundaries contains two elements.

```
PLOT.rainbowStyle = "blue/white/red"
color={1. 3.} # <= 1. are blue; above 3. red
color={3. 1.} # >= 3. are blue; <= 1. are red
```

- link enforce square shape (1:1 aspect ratio) of each cell, overrides PLOT. Yratio.
- comment = S\_xyXYtext

this option allows to draw one line of text along all the four sides of the plot box. The string array may contain up to four strings {s x,s y,s X,s Y}:

- 1. s x: lower horizontal string, i.e. comment={"xxxxxxx"}
- 2. s\_y: left vertical string, i.e. comment={"","yyy"}
  3. s\_X: upper horizontal string, i.e. comment={"","","XXXXXX"}
- 4. s\_Y: right vertical string, i.e. comment={"x","y","X","YYY"}

This option may be used to draw amino acid sequence around a contact plot box or a dot plot box.

• transparent= R 2range option allows you to make a certain range of matrix values invisible. If  $R_2range[1] < R_2range[2]$ , the specified range will be excluded from the plot, while the values beyond the range will be shown. If R 2range[1] > R 2range[2], the specified range will be shown by color, while the values beyond the range will be excluded.

Example:

transparent= $\{1. 3.\}$  # values WITHIN the range are not shown transparent= $\{3. 1.\}$  # values OUTSIDE the range are not shown

Data can also be transformed and clamped with the Trim() function.

```
read matrix "def.mat"
PLOT.rainbowStyle = "blue/white/red"
plot area def display # min/max = {-3.,17.}
plot area def color = { 0., 20.} display
plot area def color={-0.,15.} transparent={-10.,5.} display
plot area def[1:12,1:10] link display comment={"X","Y axis"}
#
#
N=210
M=Matrix(N N)
for i=1.N
  M[i,?]=Sin((Power(i-12.1 2)+Power(Count(N)-12.1 2)))
endfor
plot area M link display
     # just a nice test, default boundaries are used
read pdb "lcrn"
MDIST=Distance(Xyz(a_//ca))
s=String(Sequence(a_1./A) )
PLOT.rainbowStyle = "blue/rainbow/red"
                    # contact map for 1crn, values below 4.8 and
                    # above 10. A are not shown
plot area MDIST area color = \{4.5, 15.\} transparent=\{10., 4.8\} \
```

display link grid comment=s//s

# 2.20.53. print

print arg1 arg2 arg3 ...

The arguments may be variables or constants of integer, real, string, logical, iarray, rarray, sarray, matrix, sequence, or alignment type.

Examples:

print "no. of atoms=", i\_out, "GRAPHICS.wormRadius=", GRAPHICS.wormRadius

# 2.20.54. printf

a family of three functions for the formatted print:

- printf s\_formatString args ... # prints to stdout and s\_out
- sprintf [append] s\_formatString args ... # prints to s\_out only
- fprintf [append] s\_file s\_formatString args ... # write to a file

printf s\_formatString arg1 arg1 arg2 arg3 ...

formatted print, mostly follows the C-language printf syntax. The arguments may be variables or constants of only integer, real, string type.

*s\_formatString* may contain

- plain characters that are directly reproduced
- **ambiguous characters**: \\ backslash, \" double quote, %% percent
- escape sequences for more tricky characters (|a bell, |b backspace, |f formfeed, |n newline, |r carriage return, |t horizontal tab, |v vertical tab) and
- conversion specifications for each argument of the printf command. Each specification starts from % and may be followed by sign for left adjustment, and precision specification (e.g. %–5.2f).
- %c unsigned character
- %s string
- %d %D integer
- %[-] *i1.i2f* float (real) in decimal notation
- %g %G real in either f or e style, precision specifies the number of significant digits.
- %e %E real in [–]d.ddde+dd style
- % o % O unsigned octal
- %u %U unsigned decimal
- %x %X unsigned hexadecimal

The output is directed to the screen and is also saved in the  $s_out$  string which can be later written or appended to a file.

```
printf "Resol. = 4.1f N_m = -3d n", a, n
write append s_out "log" # append to the log file
```

See also: sprintf [append] [s\_] (prints to the s\_out string by default) fprintf [append] s\_file (directly prints to a file).

# 2.20.55. print image

print image [ window= I\_xyPixelSizes]

print the current screen image to the printer defined by the s\_printCommand ICM string variable. Use option window= to increase the resolution (however in this case bear in mind that the lines will get thinner and labels smaller). Be kind to your printer and color the background white (e.g. Ctrl-E). See also: write image s\_printCommand, View (window).

Example:

```
nice "4fgf"
color background white # or press Ctrl-E
print image
# or
s_printCommand = "lp -c -ddepartmentalColorPrinter"
print image window=View(window)*2 # increase resolution two-fold
```

# 2.20.56. quit

quit

Terminates ICM session.

# 2.20.57. randomize

# randomize internal variables in molecules

randomize vs\_ r\_angAmplitude

randomly distort current values of specified variables with either specified or default amplitude in degrees for angles and in Angstroms for bonds. The range is [CurrentValue  $- r_angAmplitude$ , CurrentValue  $+ r_angAmplitude$ ]. Default amplitude is defined by mcJump ICM-shell variable (30.0).

# randomize variables in range

randomize vs\_r\_angMin, r\_angMax

assigns random values within specified range to selected variables.

# randomize atom positions

randomize as\_ r\_amplitude

translates the specified atoms as\_randomly and isotropically according to Gaussian distribution with the specified sigma.

# randomize molecule positions

randomize ms\_ r\_amplitude

translates and rotates the specified molecules *ms*\_ randomly and isotropically according to Gaussian distribution with the specified sigma. We call it a Pseudo–Brownian random move. The same moves are used in the montecarlo docking protocol.

Examples:

```
randomize v_//!omg 50.   
# distort all variables with   
# 50 degrees amplitude randomize v_/14:21/phi,PSI -70., -50.   
# range [-70.,-50.]
```

(Note use of PSI torsion in the last example.)

# 2.20.58. read

read stuff from a disk file, pipe or string.

ICM offers several ways of reading information in:

# read from file

read ... s\_fileName

reading from a file. Just say way what and from what file. The file name is a string and must be quoted. Usually, the extension can be omitted if it is standard. Also, in several cases the program will try to find the requested file in a special directory (s\_pdbDir for a PDB file, s\_xpdbDir for an xpdb object, etc.), if is not found in the current one.

Examples:

```
read pdb "lcrn"
    # s_pdbDir will also be searched.
    # It will also read "lcrn.brk.Z"
    # you may specify file extension explicitly
read iarray "a.a"
```

# read from string

```
read ... input= s_bufferString [ name= s_newName ]
```

reading from an ICM string. Replacing file by a string is useful in CGI scripts, because the input information is easily accessible as an ICM string. Option name = *s\_newName* allows to specify a name of the new ICM-shell object.

Examples:

```
s_mat="1 2\n3 5\n0 6"
read matrix input=s_mat name="m23" # matrix m23 is created
s_seq = "> a\nAFSGFASG\n> b\nQRWTERQWTE\n"
read sequence input=s_seq # read sequences a and b
show a b
```

# read through filters: assign action by file extension.

#### read ... s\_compressed\_or\_encoded\_files

of any type directly. The files will be uncompressed on the fly, if the file extension and the corresponding filtering command are found in the the FILTER table. ICM understands .gz(gzip), .bz2(bzip2) and .Z(compress) compression.

Examples:

```
read object "aa.ob.gz"
read pdb "/data/pdb/pdb1crn.ent.Z"
```

#### read all

read all s\_allFileName

reading from a mixed file containing several ICM-shell objects (including tables) or data types. Legal types and separators:

- #>i integer\_name
- #>r real\_name
- #>s string\_name
- #>llogical\_name
- #>p preference\_name
- #>I iarray\_name
- #>R rarray\_name
- #>S sarray\_name
- #>M matrix\_name
- #>seq sequence\_name
- #>prf profile\_name
- #>ali alignment\_name
- #>m map\_name
- #>g grob\_name
- #>T table\_name # the column layout
- #>coltable\_name # the column layout
- #>db table\_name # the database layout
- #> brk # a protein-data-bank file content
- #> var # internal variables (torsions, angles, bonds) for the current ICM-object

# Example:

read all "a.all" # the file is given below

The a.all file may look like this:

```
#>r lineWidth
    1.00
#>R box4
    0. 0. 1. 1.
#>s tt.h
this is a header string of table tt. The arrays follow.
#>i tt.n
15
#>T tt
#> name bd nlines
icm 1985 160000
bee 1998 100000
inet 2000 80000
```

Such a file can be created with the

write append *icmShellObject file.all* 

command

# read index table

read { sequence | mol | mol2 } T\_selectedEntries

extract database entries selected via index table expression, e.g.

```
read index "/data/inx/SWISS.inx"
read sequence SWISS[2:15]
read sequence SWISS.ID ~ "IL2_*" | SWISS.ID == "ML2_HUMAN"
# or
read index "NCI3D"
read mol2 NCI3D.DE ~ "^benz*"
```

See the readMolNames sarray for details on database compound name storage conventions.

Index file contains an integer position of the first character of an entry (ST as in STart), and the entry length (LE as in LEngth). Accepted types of the database index files are single files with multiple entries:

```
#>s Swiss.DIR
/data/swissprot/seq
#>s Swiss.EXT
.dat
#>T Swiss
#>--ID------ST----DA-----LE-
104K_THEPA 0 906 1094
..
```

# read ftp

read .. "ftp-path/file"

reading directly from **ftp** port. The ICM can read not only from files directly accessible from your computer but also files from remote locations via ftp. ICM includes a simple FTP client to simplify access to the databases on the internet. Files names may be specified as an ftp style URL:

ftp://[ user [: password ]@] hostname [: port ]/ path/ file

If the password portion is omitted, the password will be prompted for. If both the user and password are omitted, anonymous ftp is used. In all cases passive (PASV) ftp transfers are used. If port is omitted, standard port (:21) is used.

Example:

```
read sarray "ftp://ftp.rcsb.org/pub/pdb/data/structures/divided/pdb/"+\
                "ab/pdb1ab1.ent.Z"
read sequence "ftp://emb1-heidelberg.de/toby/ph.seq"
```

URL-header may be used in existing mechanism of access to PDB:

```
s_pdbDir ="ftp://ftp.rcsb.org/pub/pdb/data/structures/divided/pdb/"
pdbDirStyle = "ab/pdblabc.ent.Z"
read pdb "lcrn"
```

Remote files are stored in your local s\_tempDir directory. Do not forget to delete them from time to time. The system table FTP can be configured to delete temporary files and deal with firewalls.

#### read http

read .. "http-path/file"

reading from http port via lynx.

Example:

read pdb "http://www.pdb.bnl.gov/pdb-bin/send-pdb?id=1crn"

\*

read unix

read .. unix unix\_command

reading from a unix pipe. (Note that you can read unix shell variables directly with the  $Getenv(s_varName)$  function).

```
read unix date
if(s_out[1:3]=="Sun")print "Go to church"
read column unix grep "^DY" fl.ou | awk '{print $11, $12}'
show def
```

# read unix cat

read .. unix cat

reading from a buffer pasted with the mouse is a special case of reading from a unix pipe. Basically, just mark anything ICM-readable in any window, paste it to your ICM session and press Ctrl-D. Note that a file name which is usually used to name the ICM-shell object is missing now, therefore it may be named 'def' (i.e. default), rename it afterwards.

Examples:

```
read alignment unix cat
 cd59n LOCYNCPNP--TADCKTAVNCSSDFDACLITKAG-----LOVYNKCWK
 ly6n LECYQCYGVPFETSCP-SITCPYPDGVCVTQEAAVIVDSQTRKVKNNLCLP
 ^D
 show def
 rename def cd_ly
 read sequence unix cat
> cd59
 LOCYNCPNPTADCKTAVNCSSDFDACLITKAG
 LOVYNKCWKFEHCNFNDVTTRLRENELTYYCCKKDLCNFNEOLEN
 ^D
# read unix cat or read string are two equivalent ways to
# load text to the s_out string
 read string
 This is the text which will end up
 in you s out string.
 ^D
# read a mixed, read all -type, input and create two ICM-shell variables:
read all unix cat
#>s ss
strrr
#>i aa
234
^D
```

# read alignment

```
read alignment [ pir | msf ] [ s_aliFileNameRoot ] [ name= s_aliName]
```

read alignment file in a natural, pir or msf formats. Upon reading, all the sequences are created as separate ICM-shell objects. The alignment is created as a separate object for msf-formatted files. In the case of other formats the alignment object is created if lengths of all the sequences together with dashed ("---") insertions are equal to each other.

# read color

read color s\_clrFile

If you want to have an alternative color file (say, "icmw.clr"), you can reread the colors.

# Example:

read color "icmw"

# read comp\_matrix

```
read comp_matrix [ s_cmpFileNameRoot ]
```

reads cmp-formatted file (\*. cmp) containing one or several residue comparison matrices.

# read conf: conformations from file

```
read conf [ i_stackConf ] [ s_stackFileNameRoot ]
```

reads and sets one specified conformation from the conformational stack file \*.cnf. If *i\_stackConf* is omitted the best energy conformation is extracted. See also read stack.

# read csd

read csd [ s\_csdFileNameRoot [ s\_csdJournalFileName ] ] [ i\_NofObjectsLimit [ i\_startingObject ] ]

reads the output of the Cambridge Structural Database (CSD) search utility, namely, FDAT-formatted file (\*.dat) and the optional session journal-file (\*.jnl). Information about atomic coordinates, connectivity, parameters and symmetry of crystallographic cell is taken from the FDAT file. The journal file contains information about chemical names of compounds. If not provided, the REFCODE csd-name is assigned to the compound name of the ICM-object. (See also Name ([ $os_{,}$ ] real)). Optional *i\_NofObjectsLimit* and *i\_startingObject* arguments allow you to extract a subset of several objects from a certain position of a multi-entry file. You can loop through all the objects by reading the chunks of up to about 1000 objects by doing the following:

The object created is not of the ICM-type, use convert or write library to create an object or an ICM-library entry, respectively. Note that you can also read compressed CSD files (see FILTER).

Examples:

To see how to generate all the symmetry-related molecules in the cell, see the transform command.

# read database

read database [ field= S\_fields ] [ group [name= s\_tableName ] ] s\_databaseFileName

read a text database with strings and numbers and create appropriate arrays. The field names in the database become names of the arrays upon reading. The list of array names will be stored in s\_out. Option group indicates that a table should be formed (or ICM-shell structure) of the constituent arrays. This table will be renamed if option name is specified.

You may also group arrays of the database to form a table with a separate command. That will allow you to sort all the arrays and search all the fields by the Find() function.

Examples:

```
read database field ={"NA","RZ"} s_icmhome+"foldbank.db" group name="tt"
read database field ={"RZ","NA"} s_icmhome+"foldbank.db" group
show foldbank
#
# ANOTHER EXAMPLE
read database "LIST.db"
show database $s_out # you may also list the arrays explicitly
write database $s_out "out.db"
```

See also: read column, write database, show database.

#### read drestraint

```
read drestraint [ only ][ s_cnFileNameRoot ]
```

read distance restraints (often referred to as *cn*) from an a . cn file. Do not forget to read drestraint types first. Option only tells the program to delete previous distance restraint settings.

#### read drestraint type

```
read drestraint type [ only ] [ s_cntFileNameRoot ]
```

read distance restraint types from a \*.cnt file. Option only tells the program to delete all previous distance restraint types settings.

#### read factor

```
read factor [ s_factorFileNameRoot ]
```

reads the Xplor-formatted structure factor file. The input is free-field, and each reflection record may be extended over several lines.

```
INDEx 1 2 3 FOBS=9.0 SIGMA=3.3 Phase=50.0 Fom=0.8
INDEx 2 -3 1 FOBS=31.0 SIGMA=2.3 Phase=20.0 Fom=0.3
INDEx 5 6 6 FOBS=44.0 SIGMA=2.0
```

To read the ICM-formatted structure factor table, just use the read table command. ICM will recognize the file type.

#### read grob

```
read grob [ s_groFileNameRoot ]
```

read graphics object from a file. To avoid conflicts with existing names of ICM-shell variables, prefix g\_ is added to the *s\_groFileNameRoot*. The default file name root is "def". Allowed input formats: ICM ( \*.gro ) or wavefront (\*.obj).

Examples:

read grob "icos" # load icosahedron from icos.gro file display g\_icos # usually g\_ prefix is added to the file name

#### read iarray

read iarray s\_iarrayFileName [ name= s\_IName]

read integer array from a file. File format is free.

#### read index

```
read index s_indexTableFile [ database= s_newDataBaseFileName ]
```

read the index file for quick access to a database. The optional argument allows to access the database file at a location different from those specified in the course of indexing with the write index command.

Examples:

#### read library

read library [ s\_libraryFileNameRoot ]

reads the ICM library files:

- icm.res and user residue libraries.
- icm.bbt bond bend angle bending and improper torsion deformation parameters

- icm.bst bond stretching parameters
- icm.cod atom codes and types
- icm.tot torsion angle energy parameters
- icm.hbt hydrogen bonding parameters
- icm.hdt surface-based hydration parameters
- icm.cmp residue comparison matrix(es)
- icm.cnt distance restraint types (cn)
- icm.vwt van der Waals energy parameters
- icm.rst multidimensional variable restraints zones

The default library name and path are defined by the s\_lib string ICM-shell variable.

# read library mmff

```
read library mmff [s_libraryFileNameRoot]
```

reads the following additional library files for the mmff94 force field:

- mmff.bbt
- mmff.bst
- mmff.tor
- mmff.tot
- mmff.vwt

To calculate the mmff energy one needs to assign atom types, and charges. The force field is switched with the ffMethod preference. An example:

Example:

```
build string "se nter his cooh"
read library mmff
set type mmff
set charge mmff
display
minimize cartesian
```

# read map

read map [ reverse ] [ s\_mapFileNameRoot ] [ name= s\_mapName ]

read ICM-electron-density map file and create an ICM-shell variable of the map type.

Option \*reverse changes the *endian* for binary maps generated outside ICM under a different operating system.

#### Reading many maps at once

```
read map [ reverse ] [ s_mapFileNames ] [ name= s_mapNames ]
```

read multiple files specified in comma-separated string (e.g. "./map/gc,./map/ge") and rename the maps by matching names from a comma-separated string. Examples:

read map "gcl,gel,gh1" "m\_gc,m\_ge,m\_gh"

read map "./gcl,./map/gel,./ghl" "m\_gc,m\_ge,m\_gh"

#### read matrix

read matrix [ s\_matrixFileNameRoot ] [ name= s\_MName ]

read ICM-matrix file and create an ICM-shell variable of the matrix type.

#### read mol

read mol [ exact ] [ s\_FileNameRoot ] [ number = { i\_number | I\_from\_to} ] [ name = s\_rootName ]

read multi-molecule MDL mol -file (a.k.a. SD-file) and create stripped molecular objects (they need further conversion). The molecules are named according to the first line of the name section of the mol/sd format. If this line is empty, the root name is taken from the option name= *s\_rootName*. If none provided the molecules are named 'm1', 'm2', 'm3',..., sequentially. If possible readMolNames is utilized.

In the default mode a pattern of single and double bonds is interpreted in order to identify aromatic systems. Then appropriate bond types are changed to aromatic (hit Ctrl-W to see the effect). This aromatic system assignment, however, is irreversible. If you write mol after that the new bond types will be saved.

Set l\_readMolArom to no if you do not want to assign aromatic rings upon reading. (and formal charge and bond symmetrization for CO2, SO2, NO2or3, PO3 ). To suppress suppress the symmetrization and consequential charging of CO2, set the l\_neutralAcids to yes .

S\_out contains all properties: All the property fields specified in the mol file, e.g.

<logp> 2.344 <cas> 234

will be stored in the S\_out array (one string for each object). The string can be further split into fields to extract the values, e.g.

cas = Trim(Field(S\_out,"cas\_rn",1,"\n")) # sarray of cas numbers logp = Rarray(Field(S\_out,"logp",1,"\n")) # rarray of logp values

Do not forget that ICM converts all strings to low-case.

Options:

- exact: enforces the exact mol/sd format. The default reading mode is more tolerant to common format violations.
- auto: automatically assigns compound names, if the name line is missing. The name is composed of the file name root and the order number of a compound.

```
read mol "ex_mol.mol" # you may skip the extension
logP = Rarray(Trim(Field(S_out,"logp",1,"\n"))) # rarray of LogP values
build hydrogen
wireStyle="chemistry"
display a_
```

See also: l\_readMolArom, l\_neutralAcids

# read mol2

read mol2[s\_FileNameRoot]

read Tripos' Sybyl mol2 –formatted file (extension .ml2) and create stripped molecular objects (they need further conversion to become ICM-objects).

Set l\_readMolArom to no if you do not want to assign aromatic rings upon reading. (and formal charge and bond symmetrization for CO2, SO2, NO2or3, PO3 ). To suppress suppress the symmetrization and consequential charging of the acidic groups like CO2, SO3, PO3 set the l\_neutralAcids to yes . These will work only if the input files contain only single and double bonds (no aromatic types).

Examples:

read mol2 "ex\_mol2" # this example file is provided

#### read movie

read movie [ s\_movFileNameRoot ]

read ICM-movie file with the Monte Carlo simulation trajectory.

See also: display movie.

#### read movie write

read movie [ s\_movie1 ] write [append] i\_fromFrame i\_toFrame s\_movie2

a movie **editing** tool. Read ICM-movie file with the MC simulation trajectory, grab a fragment [ *i\_fromFrame:i\_toFrame* ] and append it to some other file *s\_movie2*.

# read object

read object [ s\_objFileNameRoot ] [ number= { i\_objNumber | I\_objNumbers } ]

read previously formed and saved ICM-molecular-object file. If ICM object file contains several objects, all the objects are read. If argument  $i_objNumber$  is specified only the specified object is read.

The names of the loaded objects from are stored in the S\_out array, and the number of new objects in i\_out.

See also: build command to create an object from the sequence and copy object command to copy the existing object.

Example:

```
build string "se glu" name="glu"
build string "se his" name="his"
write object a_1. "obb"
write object a_2. "obb" append
delete object a_*.
read object "obb" number=2
S_objNames = S_out
show a_$S_objNames[1].
```

# read pdb

read pdb [ charge ] [ all ] [ sstructure ] [ s\_pdbFileNameRoot [ .mol/res1:res2/at1,at2,..] ]

read pdb-formatted file and create a molecular object. You can read all the information from the file or only the part you need to save program memory:

- the whole object: read pdb "/data/pdb/2ins"
- one or several chains: read pdb "/data/pdb/2ins.a,b/" (if chain is not named, refer to it as 'm')
- chain fragment: read pdb "2ins.a/3:16"
- certain atoms: read pdb "2ins./3:17/ca,c,n" (you may use name patterns with wildcards too)

Structures determined by NMR are usually represented by several models separated by MODEL and ENDMDL fields. By default only the first model will be read in.

Option all may be used to load all NMR models. Each model will be placed into a separate object. Object names will be automatically generated.

#### Error detection.

ICM detects chain missing residues according to the differences between SEQRES sequence and the residues with coordinates and returns the total number of missing residues in the *i\_out* system variable. E.g.

```
read pdb "lamo.a/"
make sequence a_1.1 # sequence lamo_1_a extracted
if(i_out>1) then
  read pdb sequence "lamo" # sequence lamo_a read
  a=Align(lamo_a lamo_1_a)
  build model lamo_a a_1.1 a # patch the missing fragments
endif
```

See also: convert command to turn it into an ICM-molecular object and the FILTER preference to see how to read the compressed pdb-files directly.

#### The fields parsed by ICM.

ICM parses most of the information from the PDB database entry and allows to manipulate with this information in the ICM-shell. The following fields are parsed:

- ATOM : all atom properties including alternative chains. To show the info: show a\_//\*. Function to extract the atom properties:
  - ◆ Bfactor: b-factors
  - Charge: charges
  - ♦ Occupancy: charges
  - ♦ Name: atom names

You can also select by many different properties of atoms, residues, molecules and objects directly in the selection expression or via the Select function.

- HETATM : all properties including alternative chains
- EXPDTA : assigned as the ICM-object type. ICM function: Type( *os*\_).
- REMARK 2: resolution is extracted. ICM function Resolution( $a_{-}$ ).
- REMARK 4: is shown as info upon reading.
- REMARK 800: description of SITEs is extracted. Can be viewed by show site. You can select these sites by a\_/F" *siteID*"
- COMPND : assigned to the object comment field. Editable and reassignable with the set comment. The comment is returned by the ICM function Namex . You may directly select with the a\_"searchString". expression.
- SSBOND:
- DBREF: database reference information shown upon reading
- SITE : sites can be shown with the show site, can be selected with the a\_/F expression.
- HELIX : returned with the ICM Sstructure function.
- SHEET : returned with the ICM Sstructure function.
- SEQRES: this sequence can differ from the sequences extracted from the ATOM records. It is read with the read pdb sequence command and becomes an ICM-shell sequence
- SCALE, TVECT, MTRIX: read but not used, the CRYST1 and ORIGX information is used instead.
- CRYST1,ORIGX : the transformation vector is returned by the ICM function Symgroup and can be applied with the transform command.

**Treatment of water molecules.** Water molecules become molecules named sequentially w1, w2, w3... Their original numbers which are stored in the residue field become their 'residue' numbers, e.g. to select water molecule number 225 and 312, do not use the w.. names of water molecues, but use the  $a_w*/225$ , 312 selection instead.

Option charge tells the program to load atomic charge from the occupancy field and reset occupancies to 1., and atomic radii from the B-factor field.

Option sstructure tells the program to automatically assign the secondary structure if it is not provided in the PDB entry.

The file will be first searched in the local directory. Extensions \*.pdb and \*.brk will be tried unless explicitly specified. If not found the s\_pdbDir directory or directories will be looked up according to the pdbDirStyle preference. This preference allows file names like pdblabc.ent recognized by the read pdb "labc" command.

read pdb "lcrn"	<pre># lcrn.brk should be either in the local # directory or in s_pdbDir one</pre>
read pdb "2ins.a/"	# load only chain 'a'
read pdb "2ins.a//ca,c,n	" # load only the backbone of chain 'a'
read pdb "lcrn./4:17"	<pre># load only 4:17 fragment from 1crn.brk</pre>

#### read pdb sequence

read pdb sequence [ resolution ] [ s\_pdbFileNameRoot ]

quickly extract only amino-acid sequence from SEQRES records of a pdb-formatted file without actually loading molecules.

It is important to understand that sometimes sequence from the SEQRES records **does not match** the sequence extracted from the ATOM records, because some residues in flexible loops and ends are invisible. Option resolution appends X-ray resolution to the sequence name (like 9lyz\_a19, 19 stands for 1.9 resolution). 'No' is appended for NMR and theoretical structures. It can be used later by the group sequence unique command to compile the representative list of PDB chains.

PDB is famous for having numerous errors which are never fixed. In SEQRES sometimes the stated number of amino-acids in SEQRES does **not** correspond to the actual number of amino-acids (e.g. 1cty, 1ctz, 1ctz, 2tmn, 1ycc, 2ycc).

The sequences will be called according to the pdb code and the chain name. In case of one chain without a name, ICM assigns name "m" . e.g. 1est\_m , 2ins\_a , 2ins\_b.

Records are converted to lower case. In rare cases, such as lfnt, in which there are both upper and lowercase chain names, the lowercase names become uppercase, e.g. lfnt\_a for the first chain and lfnt\_A for the 33-rd chain.

An example script to detect problems with pdb sequences (you can build the list with the makeIndexPdb and mkUniqPdbSeqs macros)

```
read sarray s_pdbDir + "pdb.li" name="a"
l_info = no
errorAction = "none"  # otherwise breaks at pdblaa5.ent
for i=1,Nof(a)
  read pdb sequence s_pdbDir + a[i]
  delete sequence
endfor
Error> no SEQRES records in file /data/pdb/af/pdb0af1.noc.Z
Error> no SEQRES records in file /data/pdb/ao/pdblao2.ent.Z
Error> no SEQRES records in file /data/pdb/ao/pdblao4.ent.Z
Warning> Sequence of chain "pdblati_c" starts with 'UNK' and is unknown
Warning> Sequence of chain "pdblati_d" starts with 'UNK' and is unknown
```

# read profile

read profile [ s\_prfFileNameRoot ] [ name= s\_prfName]

read ICM-sequence profile from a file and create an ICM-shell variable of profile type.

#### read prosite

read prosite [ s\_prositeFileName ]

read all the patterns from the prosite database (Amos Bairoch, University of Geneva, Switzerland) and create two string arrays: prositeNames, and prositePatterns, containing names and patterns, respectively. The search may be performed by the find prosite command. Check also the find prosite command.

Examples:

See also: s\_prositeDat.

#### read rarray

read rarray [ s\_rarrayFileNameRoot ] [ name= s\_RName]

read real array from a file. File format is free.

#### read sarray

read sarray [ s\_sarrayFileName ] [ name = s\_SName ]

read any text from a sar-file as a bunch of strings separated by carriage returns. Create an ICM-shell variable of sarray type.

#### read segment

read segment [ s\_fileName ]

reads a simplified description of protein topology from a file. You can append your description to the provided foldbank.seg file.

Examples:

read segment s\_icmhome + "/foldbank"

See also: assign sstructure segment, find segment, write segment.

#### read sequence

read sequence [ { fasta | swiss | pir | gcg | msf ] } [ s\_seqFileNameRoot ] [ field= S\_ ]

read amino-acid or DNA sequence from a variety of sequence file formats and create an ICM-shell variable of sequence type.

See also: swissFields

#### read sequence database

read sequence T\_indexSubset

read amino-acid or DNA sequence from an indexed sequence database. *T\_indexSubset* contains the selected entries which can be defined by a table expression (e.g. SWISS.ID=="^IL2\_\*"). The names of the sequences extracted from the database to the ICM memory are stored in the S\_out system string array. i\_out contains the number of the sequences loaded. These variables are used in automated scripts for bioinformatics (see searchSeqDb or searchPatternDb) macros.

Examples:

```
read index s_inxDir+"/SWISS" # load the Swissprot index
read sequence SWISS[1:20]  # first 20 entries
show S_out[1], $S_out[1]  # show the 1st name and the sequence
#
read sequence SWISS.ID=="^IL2_*" SWISS.ID!="*_MOUSE"
S_seqNames = S_out
for i=1,Nof(S_seqNames)
seqName = S_seqNames[i]
show seqName, Nof(String($seqName),"[KR]") # stat. of positive charge
endfor
```

# read stack

read stack [ append ] [ s\_stackFileNameRoot ]

read stack of conformations from a cnf-file. This command resets the energy terms as they were saved in the cnf-file. The terms string is returned in the s\_out variable.

# read string

```
read string [ s_textFile ] [ name= s_sName]
```

read any text from either standard input or a *s*\_*textFile*. Place the result into the s\_out string. Reading string from standard input can be used to get URL–encoded stream generated by the HTML–form. The read string command can also read from ftp.

See also: read unix command which allows to read in ICM the output of any unix command.

In the last example the file will be downloaded from the PDB site and dumped into the pdbList string variable.

# read table

```
read table [ database ] [ name= s_tableName ] [ s_tableFileName ]
```

ICM needs two lines with the table name and the field names in the following format: (an example):

```
#>T atm
#> name code weight
hydrogen 1 1.008
```

To read a \*.csv or \*.tsv formatted files, use the read column group command.

See also: table, icm.tab file.

Examples:

```
read table s_icmhome+"atm" name="ATOMS" # atm.tab file by default
sort ATOMS.weight # sort according to the weight array
```

```
read column [separator= s_Separator] [group [name= s_tableName]] s_fileName
```

read a multicolumn table with strings and numbers and create appropriate arrays. If you add a ruler starting from #> and looking like this

#>-name1---name2----name3-----name4---

the arrays will be created with specified names. If ruler is missing, default names (I1, I2 ..., R1, R2,...,S1, S2, ... for iarrays, rarrays and sarrays, respectively) will be created. You may control field formation by  $s\_fieldDelimiter$  variable or by adding **separator** =  $s\_Separator$  explicitly. The list of array names will be stored in  $s\_out$  so you can always say

```
read column "res"
show column $s_out
```

#### Reading comma-separated-value or tab-separated-value formats

To read a table in comma-separated-value (*csv*) or tab-separated-value (*.tsv*) format redefine the s\_fieldDelimiter value and use the read column group command.

read column group name="t" "t.csv"

See also: write column, show column, icm.col.

```
read variable [ s_varFileNameRoot ]
```

read ICM-molecular object variable values (torsion angles, phase angles, bond angles, bond lengths) from a var-file.vs\_out selection will contain a selection of variables which have been modified by the command. Variables are assigned according to the residue number and the variable name. If residue name is different (i.e. you want to assign phi,psi of an alanine 15 to glycine 15), the program sends a warning. If more than one molecule is present in the current object, matching of molecule names is required. See also set vs\_command.

read view [ s\_viewRarrayName ]

read rarray of 36 display parameters for window size, scale, view matrices, etc. and set them. See also: set view, View() function

Examples:

```
build string "se ala"
display
write View( ) "a"
# rotate the image
read view "a" # restore view
```

read vrestraint [ s\_rsFileNameRoot ]

read variable restraints (often referred to as rs ) from a \*.rs file. Do not forget to read vrestraint types first. Option only tells the program to delete previous variable restraints.

```
read vrestraint type [ s_rstFileNameRoot ]
```

read variable restraint types from a \*.rst file. Option only tells the program to delete previous variable restraint types.

# 2.20.59. rename

rename oldName { s\_newName | u\_newName }

rename anything to anything else. More specifically you can rename commands, ICM-shell variables, objects, molecules, residues and atoms. Renaming commands is possible, but then you must not forget to change them in all the standard ICM-scripts. Using aliases instead allows you to use both the original and the translation, however it slows down the ICM-shell interpretation. Be careful with a new name to avoid name conflict.

# 2.20.60. rename object

```
rename { os_[full]|ms_|rs_|as_} s_newName
```

change selected names. To change the long name of the object (it can contain space in contrast to a regular object name), use the full option.

Examples:

# 2.20.61. return

```
return [ error ] [ s_message ]
```

return from a macro before endmacro usually under specific conditions. Similar to exit command returning from a file to interactive mode. Option error will set the error flag which can later (outside the macro) be checked with the Error() function. The message *s\_message* will be stored in the s\_out string shell variable.

Examples:

```
macro aa
    if(Nof(sequence)==0) return # a silent return
    ....
endmacro
macro aaa as_1
    if(Nof(as_1)==0) return error " aaa> Nothing to do"
    show as_1 # a pretty silly macro
endmacro
macro bbb
    if(Nof(object)==0) return error " Error_in_bbb> No objects in the system"
    ....
endmacro
bbb # call this macro
    if(Error) print "something went wrong with macro bbb"
```

# 2.20.62. rotate

the main rotate command. Subtypes of this command include rotate object, rotate view, rotate grob.

# rotate object

rotate [ os\_ | ms\_ | g\_grob ] M\_rotation

rotate an object (os\_), one/several molecules (ms\_) or g\_grob with the specified rotation matrix.

Examples:

# rotate grob

rotate g\_grobName M\_rotation

rotate a graphics object.

Examples:

```
read grob "oblate"
display g_oblate magenta
rotate g_oblate Rot({0. 0. 1.},30.)
```

#### rotate view

rotate view M\_rotation

rotate view in the graphics window with respect to the screen axis X (horizontal), Y (vertical) and Z (perpendicular to the screen). This command is great for creating movies or demos when the graphics should be manipulated from a script.

Example:

```
build "alpha"
read movie "alpha"
display a_//ca,c,n
for i=1,100
    load frame i
    rotate view Rot({0. 1. 0.} , -1.) # rotate around Y by -1 deg.
endfor
```

See also: the View() function and the set view command.

# 2.20.63. set family of commands

to change properties of existing icm-objects, e.g.

set bfactor  $a_//c*$  20.

# set area sequence : positional factors for sequence alignment

```
set area seq_[ { R_factors | r_factor } ]
```

sets/resets a property array assigned to a sequence. Each amino acid can be assigned a relative solvent accessibility value for this residue in a three–dimensional model. 0. – fully buried (the highest possible factor), 1. – fully exposed. These values can also be used to influence the alignment (buried residues with accessibilities close to zero will have larger contributions). The exact dependence residue–residue score

factor on this value is defined by the accFunction array.

Example:

set area lcrn\_m 0.
set area lcrn\_m Random(0. 1. Length(lcrn\_m))

```
See also: accFunction, Align(seq1 seq2 area)
```

### set atom

```
set as_singleAtom [ { R_3Dvector | as_select } ]
```

set as\_manyAtoms M\_XYZ

With a single atom selection, ICM sets a given atom to the center of gravity of the corresponding molecule (no arguments), given point in space ( $R_3D$ vector argument) or center of gravity of selected atoms ( $as_select$  argument).

If multiple atoms are selected, ICM sets the specified atoms to their new XYZ positions. The XYZ matrix can be returned by the Xyz (as\_) function.

Examples:

```
build string "se ala his glu"
set a_/3/ca a_//ca # 3rd Ca to the center of mass of all Ca s
set a_/3/ca {-3., 12., 14.5}
set a_//vt1 # set the first virtual atom to the center of mass
randomze a_//vt1 0.1 # randomize the vt1 position in case of singularity
```

For ICM molecular objects, in the most popular operation (set  $a_1//vt1$ ) the first of the two virtual atoms (vt1) attached to the beginning of the selected molecule is set to the center of gravity of the same molecule. The purpose of this action is to simplify molecular rotation and translation via the first six free virtual variables. The tvt2 and tvt3 torsions and avt2 planar angle determine rotation of the whole molecule around the axes passing through the center of gravity. Useful for docking.

Examples:

See also:

command	description
set a_/* s_secondaryStructure	to change phi,psi angles according to secondary structure,
virtual atoms/variables	information about virtual atoms and variables and the
move	command which goes further and actually changes the topology of the ICM-tree.

# set bfactor

set bfactor as\_ { r\_NewFactor | R\_NewFactors } : \*set \*bfactor rs\_ R\_NewResidueFactors

set B-factors of selected atoms to a specified real value, respectively. To assign individual b-factors, provide a real array with b-factors for each atom. To assign the individual b-factors at the residue level, provide matching residue selection and *R\_NewResidueFactors* array.

Examples:

```
buildpep "ala his trp" # also includes N- and C- terminal groups
set bfactor a_//* 20.
set bfactor a_//ca {20.,10.,30.} # individual atomic factors
set bfactor a_/2:18/ca,c,n 10.
set bfactor a_/* {10.,20.,30.,20.,10.} # individual residue factors
```

# set bond type

set bond type as\_class1 [ as\_class2 ] { i\_type }

set the bond chemical type (0 – undefined, 1 single, 2 double, 3 triple, 4 aromatic, 9 quadruple, 10 amid).

set bond auto ms\_

with the auto option the command automatically reassigns patterns of single and double bonds. It performs the following operations:

- identify aromatic rings in object os\_ from patterns of single and double bonds. Use preference wireStyle = "chemistry" (Ctrl-L) to see the bond types. This is done automatically upon reading of objects, mol and mol2 files if logical l\_readMolArom is set to yes.
- for ICM objects, set ICM bond variable types according to bond chemical type, atom types and distance between them

Example:

```
read pdb "lcrn"
display
wireStyle="chemistry"
set bond type a_//c a_//o 2 # double  # standard bonds in a/acids
set bond type a_/phe,tyr,trp/[cn][gdez]* | a_/arg/cz*,nh* 4 # aromatic
set bond type a_/as?/cg*,od,od1 | a_/gl?/cd*,oe,oe1 2
```

build hydrogen a\_/A

#### set charge

set charge as\_select { r\_NewCharge | add r\_Increment }

sets or increments partial electric charges of selected atoms to or by specified real value, respectively.

set charge as\_select { R\_NewChargeArray | add R\_ArrayOfIncrements }

sets or increments partial electric charges of selected atoms to or by a specified real array. The array assignment is useful for saving and restoring the charges.

Examples:

set charge a\_//\* 0.
set charge a\_/lys/nz | a\_/arg/cz 1.0
set charge a\_/asp/od\* | a\_/glu/oe\* -0.5
oldCrg=Charge(a\_//\*)
set charge a\_//\* 0.0
set charge a\_/asp/od\* | a\_/glu/oe\* -0.5
# do something with these simplified charges
set charge a\_//\* oldCrg

See also: set charge formal, set charge mmff.

### set charge formal

set charge formal as\_select r\_NewFormalCharge

sets formal partial electric charges of selected atoms to or by a specified real value. The charge will be rounded to the nearest value proportional to 1/12th. The following values are common: +-N, +-N/2, +-N/3, +-N/4., +-N/6. Note that the formal charge can not be arbitrarily changed without appropriate changes in the surrounding bond types. The formal charge will be considered by the Smiles function.

Example:

set charge formal  $a_//n - 0.333$  # a formal charge of -1/3.

See also: set charge, set charge mmff.

#### set charge mmff

set charge mmff as\_select

set atomic charges according to the rules described in a series of publications on the Merck Molecular Force Field abbreviated as MMFF94 or just MMFF.

This command requires the mmff atom types (see the set type mmff command). Do not be surprised that the methyl groups have zero partial charges. That is how they are defined in the MMFF algorithm.

Example:

```
set type mmff
                     # mmff atom types
show atom type mmff
set charge mmff
                     # charges
#
rinx MOL3D
            # index for small molecule database
read mol2 MOL3D[23:34]
for i=2,Nof( object )
  set object a_$i.
  display
  build hydrogen
  convert
  set charge mmff
  display ball
  color a_//* Charge(a_//*)//{-1., 1.} ball
endfor
```

See also: set charge, set charge mmff.

### set comment

set comment [ append ] os\_Object s\_comment

set a text comment string to the specified object. This annotation is preserved in the read object and write object commands.

Examples:

```
read object "crn" set comment append a_ "\n The template for modeling\n Energy minimized\n"
```

### set a flag of an alternative atom position

set comment "A" as\_alterAtoms

set alternative status to the selected atoms. The alternative flag can be read from a pdb file. This flag marks alternative geometrical positions of atoms which are described in the previous ATOM records. For example, the same side–chain or a water molecule can occupy several positions. The symbol of alternative position (usually 'a','b' or 'c' character, since ICM converts the strings to low case) precedes the residue name field. The alternative positions can also be selected with the  $a_//A$  alterChar selection.

Example:

```
read pdb "lcbn" # has alternative positions
show a_//Ab  # show alternative pos. 'b'
set comment a_//Aa "x" # rename 'a' positions to 'x'
```

### set comment to a sequence

set comment [ append ] seq\_ s\_comment

set comment to a sequence.

Example:

```
a=Sequence("AFSGDHAGSFDSGAHGSDFASGDA")
set comment a "a random test sequence"
```

#### set comp\_matrix: redefine residue comparison matrix.

```
set comp_matrix [ add ] r_increment [ s_ijPattern ]
```

change the numbers in the residue comparison matrix, called comp\_matrix by a number typically between 0. and 0.2. This may be very important for generating a reasonable alignment for sequences with low sequence similarity. The result is similar to reducing the gapOpen parameter by about 0.1.

Examples:

### set directory

set directory s\_newDirectory

change the *current working directory* from inside the icm-shell. We recommend using: alias cd set directory "\$1". In this case you can change directory in the Unix/DOS style.

Example:

```
set directory "/usr/tmp"
cd .. # uses alias from _aliases.
# cd .. is equivalent to set directory ".."
```

### set drestraint

set drestraint as\_select1 as\_select2 i\_DrestraintType

sets distance restraints of specified type between selected sets. Drestraint types (integer numbers) can be either read from a \*.cnt type file or set directly by the set drestraint type command and shown by the show drestraint type command.

Examples:

2.20.63. set family of commands

### set drestraint type

set drestraint type i\_DrestraintTypeNumber r\_WeightingFactor r\_LowerBound r\_UpperBound [
local r\_Sharpness ]

creates a distance restraints type. Drestraint types (integer numbers) can also be read from a \*.cnt type file and command and shown by the show drestraint type command.

Examples:

```
# type 11, weight 10., bounds [1.,3.]A, target dist. 1.5
set drestraint type 11 10. 1. 3. 1.5
# local type, sharpness 5.
set drestraint type 12 10. 1. 3. 1.5 local 5.
```

#### set site

set site [ only ] { [ seq\_from [ ali\_ ] ] | s\_siteString } { seq\_to | ms\_to } [ append ]

transfer (or reassign) sites (listed in feature tables of swissprot entries) from a sequence or string to a destination sequence  $seq_to$  or a selection of molecules  $ms_to$ .

If alignment is not provided, the sequences will be automatically aligned to find residue-residue correspondences and the reliability of the alignment will be reported. If the source of sites is not provided, the sites will be transferred from the sequences linked to objects. The list of sites and their one-letter codes is given below. Normally this command appends to the list of existing sites, unless the only option is given in which case the old sites are dismissed. The effort is made to avoid repetition and retain only the unique set of sites.

Alternatively, if the string is specified, create a new site according to the provided legal site string  $s\_siteString$  (e.g. "FT ACT\_SITE 15 15 Catalytic residue"). The format of the site string is the same as in the swissprot sequence entries. The list of legal site types is given in the Glossary.

The site residues in objects can be delete with the delete site command and selected with the a\_/F SiteCodes selection, (e.g. a\_/FAB selects residues involved in binging and active site).

Example:

```
read sequence "a.seq"
set site "FT ACT_SITE 15 15 active site residue" a
```

### set site to a residue selection

set site [ only ] rs\_s\_sideString

assign sites to a molecular 3D object (simpler than the previous Swissprot-like definition).

Example:

read object s\_icmhom+ "crn.ob"
set site a\_/10:13 "candidates for mutagenesis"

# set field

set field selection { r\_FieldValue | R\_arrayOfValues } [ number = i\_fieldNumber ]

set field clear selection [ number = i\_fieldNumber ]

set user-defined values to atoms, residues, molecules or objects selected. Atoms have one user-field, residues have three, molecules and objects have sixteen. To specify which field you need to set, use the number= option.

To extract the property use Field ( *selection*, *i\_fieldNumber* ) function.

Level	Max.Nof_fields		example
Atom	1	set	field a_//c* Mass(a_//c*)
Residue	3	set	field a_/trp 1. number=2
Molecule	16	set	<pre>field a_W Random(1.,10.,Nof(a_W)) number=12</pre>
Object	16	set	<pre>field a_*. Rarray(Count(Nof(a_*.)))</pre>
User defir	ned fields can furt	her be	e 2D or 3D averaged with the Smooth function and selected by with the
Select	function.		

# set font

```
set font [ auxiliary ] [ bold ] [ italic ] [ underline ] [ { atom | residue | variable |
string } ] i_Size s_FontName
```

set current font for atom-, residue-, variable-, or string-labels in the graphics window. Strings can be displayed in either their main font or the auxiliary one (option auxiliary). The following fonts: times, helvetica, courier and symbol, should be available. Default fonts are defined in the icm.clr file.

Examples:

```
set font 28 times  # 'Times' font, size 28
#
build string "se his"
atomLabelStyle="[C]"
display wire atom label
set font atom 14 bold  # for atom labels
#
set font auxiliary bold italic helvetica 28
```

# set grob coordinates and string label

```
set g_grobName M_Xyz : *set g_grobName *label [ *append ] s_Label
```

Set new coordinates to the vertexes of the specified graphics object. The matrix dimensions should correspond to the number of vertices. The initial coordinate matrix can be extracted with the Xyz (grob) function.

```
read grob s_icmhome+"beethoven" # try stravinsky if you want
display
display "DESTRUCTION OF CLASSICAL MUSIC"
xyz= Xyz( beethoven )
fuzz = Random(-0.2,0.2,Nof(xyz),Length(xyz))
xyz = xyz + fuzz
set beethoven xyz
color beethoven Random(Nof( beethoven ),3, 0., 1.)
```

# Invert grob normals

set grob reverse

change direction of vertex plane normals in *all* grobs to change direction of lighting and sign of the Volume function.

```
set g_grobName1 g_grobName2 .. reverse
```

change direction of normals in *specified* grobs. In some simple grobs the order of vertices defines the normal implicitly. In this case the order is changed.

An example in which we contour a density map, split the grob into outer shell and cavities and measure their volumes:

```
read pdb "lest.m/"
make map potential 1. Box( a_ )
make grob m_atoms 0.2 exact solid
split g_atoms
set grob reverse # invert normals of all grobs
set g_atoms2 g_atoms3 reverse # invert normals of some grobs
Volume(g_atoms1 ) # outer shell is now illuminated from inside
Volume(g_atoms2 ) # cavities have now positive volume.
64.865657
```

### set key

set key s\_keyName s\_Command

binds key to a command. Allowed keys: F1, ... F12, Ctrl-F1, ... Ctrl-F12, Ctrl-A, ... Ctrl-Z, Alt-A, ... Alt-Z. Add "\n" at the end if you want your command to be automatically executed.

Examples:

```
set key "F1" "set plane 1"
set key "Ctrl-B" "l_easyRotate=!l_easyRotate"
set key "F6" "varLabelStyle=\"nextItem\"\n"
```

### set label

set label as\_atomForResidueLabels

assign residue labels to the selected atoms as\_atomForResidueLabels .

Examples:

```
build
set label a_/tyr/cb # move label from Ca's to Cb's for all tyrosines
```

### set label distance

```
set[residue]labeldistance rs_[{R_3displVector|M_displMatrix}]
```

reset the relative displacements of the selected residue labels  $rs_{-}$  to their default of the specified positions. If vector is specified, all the relative displacements are set to this vector, if a relative displacement matrix Nx3 is given, each selected label is moved to the specified relative position. The default position is the relative displacement of {0.0.0.} from the residue label carrying atom (usually the Ca atom for peptides, also see the set label as\_command). See also: GRAPHICS.resLabelDrag

Examples:

```
build string "se tyr tyr glu als his"
set label a_/tyr/cb # move label from Ca's to Cb's for all tyrosines
display a_* residue label
GRAPHICS.resLabelDrag=yes # now drag labels with the MiddleMB
set label distance a_/2:4 # reset labels for residues 2:5
set label distance a_/2:4 {1. 0. 3.}
```

# set the current map

set map m\_theMapYouWantToWorkWith

assigns the current map status to the specified map.

```
set object [ os_newObjName ]
```

assigns the current object status to the specified object. Switches to the next one by default.

Examples:

set object a\_crn. # set it to object crn
set object a\_1. # set it to the first object
set object # switch to the next or alternative

### set occupancy

set occupancy *as\_select r\_NewOccupancy* 

sets occupancy of selected atoms to or by a specified real value between 0.0 and 1.0

Examples:

set occupancy a\_/2:5/!ca,c,n,o 0.5

set occupancy a\_/2:18/ca,c,n 1.

# set plane

```
set plane [ i_plane ] [ { off | on } ] [ name= s_planeName ]
```

toggles the specified graphics plane on and off. Up to seven planes can be set. Optional name is assigned to a plane. It is a convenient way to operate with complex composite images. Every image is assigned to a certain graphical "plane" when displayed. Different parts of the image can be assigned to different planes. For example, **plane 1** may contain wire representation of molecule1, **plane 2** its molecular surface ("surface") and **plane 3** molecule2 in "xstick" representation. It can be achieved by pressing "F2" and "F3" (which are aliased to **set plane 2** and **set plane 3**, respectively) before displaying surface and xstick respectively. Now by pressing "F1", "F2" and "F3" one can toggle these three screens (or planes) to display any combination of them. It is much better than undisplaying and displaying them directly, especially for representations requiring serious computations like surface and skin. The main modes of the set plane command:

- set plane 2 : if plane2 is 'off', make current and switch it 'on'; if it is 'on', switch it off.
- set plane 3 on : switch the plane on, but do not change the current plane
- set plane 4 name="homologue" : just assign name to the plane, no switching

#### Examples:

build string "se ala ala" # create a peptide set plane 2 # F2 with the cursor in the graphics window display surface set plane 3 # F3 with the cursor in the graphics window display xstick set plane 2 # switch off the surface set plane 2 # switch the surface back on set plane 3 # switch off the xstick set plane 3 # switch the xstick back on

# set property

set plane [ i\_plane ] [ { off | on } ] [ name= s\_planeName ]

### set sstructure backbone

set rs\_s\_SecStructPattern

assign the specified local secondary structure to the selected residues of an ICM-type object. Note that this command **changes the conformation** of the selected residues, in contrast to the command <code>assign sstructure</code>.

The phi,psi angle values are changed according to the following code:

ss_code	phi,psi angles	description
_	-179.9,179.9	extended conformation
Е	-139.0,135.0	antiparallel beta strand
e	-119.0,113.0	parallel beta strand
Н	- 62.0,-41.0	alpha-helix

G - 49.0,-26.0 G-helix (3/10) I - 57.0,-70.0 I-helix P - 78.0,149.0 poly-proline 2 helix L + 57.0,+47.0 Left-Alpha Examples:

```
build "all"
display ribbon residue labels
set a_/2:8 "HHHHHHH"
center
set a_/1:12 "HHHHHH__EEEE"
center
set a_/A String("H", Nof(a_/A) )
center
set a_/A String("_", Nof(a_/A) )
center # ONLY UNFIXED PHI,PSI VARIABLES ARE SET, SO pro IS BENT!
set a_/A String("G", Nof(a_/A) )
center
set a_/A String("E", Nof(a_/A) )
center
```

#### set sstructure to sequence

set sstructure seq\_s\_SSstring

set secondary structure *s\_SSstring* to the specified sequence. If *s\_SSstring* is an empty string, the secondary structure definition is removed.

Examples:

### set stack energy

set stack energy R\_NewEnergies

resets energy values for conformations stored in the stack . It may be useful for the subsequent sort stack command.

### set symmetry to a torsion

set symmetry {  $1 \mid 2 \mid 3 \mid 6 \mid$  exact | heavy | pseudo }  $vs_{-}$ 

assigns rotational symmetry to selected variables. This symmetry will be used to automatically transform the value of a torsion angle into [-180.0/symmetry, 180.0/symmetry] range.

Options are the following:

- exact impose exact symmetry (methyl groups=3, xi2\_phe=2)
- heavy impose exact symmetry as if there are no hydrogens
- pseudo- impose pseudo symmetry (no\_hydrogens + xi2(his,asn,gln))

# set crystallographic symmetry group

set symmetry os\_object R\_6cell s\_symgroup i\_NofMolecules

assigns symmetry and cell parameters to selected object(s).

Information is supposed to be compatible with that provided in CRYST1 PDB card:

- R\_cell should be a 6-component real array, containing values of A, B, C, alpha, beta and gamma.
- *s\_symgroup* is a string description of the space group. To check validity of the *s\_symgroup*, use the Symgroup(*s\_symgroup*)} function, which will return a number from 1 to 230 for a valid space group name. Fast Fourier transformations are currently supported for *s\_symgroups* "P 1" and "P 21 21 21", but all the other commands (make map transform etc.) will work on any space group defined in the International Tables for Crystallography.
- *i\_NofMolecules* is the Z-value, the number of polymeric chains in a unit cell.

Examples:

#### set table

set table t\_theTableYouWantToWorkWith

assigns the current table status to the specified table (similar to set object os\_ to set the current molecular object).

#### set energy or penalty terms

set terms [ only ] [ s\_termsString ]

set energy and/or penalty terms for further energy calculations. Each term has a two-character abbreviation. The terms are appended to the string unless option only is specified. The final energy-term string is returned in the s\_out string

Examples:

# vacuum terms, solvation and entropy set terms only "vw,14,hb,to,el,sf,en" set terms "tz" # add tethers to the list

# set tether

set tether [ align | ali\_ ] [ exact ] [ only ] as\_atomsToBePulled [ as\_atomTargets ]

this command sets tethers restraining atoms of ICM-object (selection *as\_atomsToBePulled*) to corresponding atoms of another object (*as\_atomTargets*). The *as\_atomTargets* selection may also contain only **one** atom, in which case all *as\_atomsToBePulled* will be tethered to a single atom. If the second argument is not specified, all the *as\_atomsToBePulled* atoms are tethered to the origin (the {0. 0. 0.} point). Option only signals that all previously imposed tethers must be deleted.

In a residue pair the only the backbone atoms such as ca,c,n,o,ha,hn are tethered with the exception of

- identical residues: all atoms are tethered
- F with Y (all but the hydroxyl)
- D with N
- E with Q

The number of imposed tethers is saved in i\_out.

See also: superimpose, alignment options, minimize tether.

Example (try it as one continuous session):

### set atom type

set type [ mmff ] [ as\_ { i\_type | I\_type } ]

assigns the specified atom type (see icm.cod or show atom type [mmff]) to the selected atoms. Both the ICM- and the mmff- atom types may be manually adjusted to correct the automated set type mmff command.

### set type sequence

set type [ seq\_ ] { protein | nucleotide }

assigns the specified type to the sequence. The type can be returned by the  $Type(seq_)$  function.

Example:

```
aaa = Sequence("AAAAATAAAA")
set type protein aaa
```

# set type mmff

```
set[atom]typemmff[os_]
```

automatic assignment of the MMFF atom types for the selected or the current object of any type. This object can be both ICM-object or a non-ICM object, provided three conditions are satisfied:

- 1. the bond types are set correctly
- 2. the formal charges are set correctly
- 3. the object is complete and has hydrogens (see the build hydrogen command)

This command is a prerequisite for the set charge mmff command.

### set van der Waals radii

set type "vw radii" I\_vwTypes R\_vwRadii

reset radii defined in the icm.vwt for *I\_vwTypes* to the *R\_vwRadii* values. The van der Waals radii are used for the surface calculation in the show surface area

command

### set electrostatic radii

set type "vwel radii" I\_vwTypes R\_vwRadii

reset electrostatic radii marked as vwel defined in the icm.vwt. The electrostatic radii are used in the boundary element electrostatic calculation.

### set 3D view rotation, tranlation and size

set view R\_36parameters

sets all the parameters of the graphics window (position, size, zoom, rotation, etc.) according to a rarray of 36 numbers. This array is returned by the View() function and can be created, read and written as an ordinary real array. Aren't you disappointed that you still do not know the meaning of these parameters? It is dull, believe me, use the command and take it easy. See also: View, rotate view.

Example:

```
display a__crn. ribbon # now move the molecule, resize window ..
write View( ) "a.view" # write 36 numbers in a file
    # again: rotate, move/resize the window etc., or quit the session
read rarray "a.view" # read 36 parameters
set view a # restore the view
```

### set vrestraint

```
set vrestraint [ energy ] rs_ [ s_rsTypeName1 s_rsTypeName2 ... ]
```

sets variable restraints of specified types to the selected residues rs\_. Variable restraint type names (strings) can be read from a \*.rst type file and shown by the show vrestraint type command. Option energy enforces the "energy" type of vrestraint.

Number of imposed variable restraints is saved in i\_out.

Examples:

```
set vrestraint a_/* \prescript{\#} assign all zones to relevant residues set vrestraint a_/ala "aa" "ab" \prescript{\#} assign alpha and beta zones to all Ala residues
```

#### set vrestraint

set vrestraint [ only ] [{ energy | fix } ] vs\_r\_l r\_2 [ r\_3 ] [ R\_values ] [ name= s\_rsName ]

impose a set of vrestraints to the specified variables  $vs_{-}$ . The zone will be a multidimensional elliptical well *around current values* (default), or the specified  $R_values$  values, of the selected variables. The shape of the well in each dimension is a *soft square well*. Three types of vrestraints can be imposed, depending on the option:

- **probability** vrestraints (the default). They are marked as "rs" in the icm.rst file. Probability vrestraints are used in the BPMC procedure to define the distribution of random steps. The well parameters are as follows:
  - $\bullet$  *r\_1* : *r\_relProbability* , the relative probability of this vrestraint
  - ◆ *r\_2* : *r\_wellRadius*, the well radius

The relative probability is in arbitrary units, it is only important as a relative number in a *group* of the vrestraints.

- energy: "Energy" vrestraints (marked as "rse" in the icm.rst file. They allow to form multidimensional wells around groups of variables and are used to softly restrict the variables to certain zones (see the "rs" energy term). The well parameters are as follows:
  - ◆ *r\_1* : *r\_energyDepth* (it must be negative for attractive wells)
  - ◆ *r*\_2 : *r\_fractionFlat*
  - ♦ r\_3 : r\_wellRadius

Parameter *r\_fractionFlat* (between 0. and 1., default 0.) defines flat fraction of the energy well for the energy vrestraints.

**Note:** one can create both **wells** and **bumps** using negative and positive values of *r\_energyDepth*, respectively Example:

```
build string "se nter ala ala cooh" set vrestraint energy v_/3/psi -20., 200., 0.2 \# WELL OF DEPTH 20. set vrestraint energy v_/3/psi 20., 200., 0.2 \# BUMP OF HEIGHT 20.
```

An example from the \_dock2mol.icm script: imposing an individual restraint for the virtual bond:

# no penalty for deviations up to 15A set vrestraint energy v\_2//bvt1 only -50.0 0.5, 30.0

- $R_values$  contains target values for each angle in the selection  $vs_v$ , e.g. {-120.,60.}, By default the target values are taken from the current values of the selected variables.
- fix: Vrestraints on "fixed" variables (marked as "rsr" in the icm.rst file). These are used to define switches between different fixed conformations, e.g. alternative conformations of sugar rings, proline rings, switches between L and D amino-acids etc. These switches will be tried in the montecarlo procedure if these variables are included in the set of vs\_MC variables but not included in the set of the minimization vs\_min variables. The parameters are defined as follows:
  - ♦ r\_1 r\_relEnergy, relative energy of a conformer
  - ♦ r\_2 r\_relProbability.

The *r\_relProbability* is in arbitrary units as for the probability vrestraints.

Example with L–D transition, through changing the sign of the two phase angles:

```
build string "se ala his trp" set vrestraint fix V_/3/fha,fcb Value( V_/3/fha,fcb ) 0. 1. name="l" set vrestraint fix V_/3/fha,fcb -Value( V_/3/fha,fcb ) 0. 1. name="d" montecarlo V_/3 v_//*
```

The radius of the vrestraint well (in degrees for angles) is given by the  $r\_wellRadius$ . Option only deletes all the previous vrestraints. The name is optional. The names of the "probability" and "fix" vrestraints are be shown in the output of the montecarlo procedure. The names need not be unique.

Example: creating a file with equal probability vrestraints around stack conformation angles with 30 deg. radius:

read stack "f1" # read conformational stack for i=1,Nof(conf) # go through all the conformations load conf i # load them one by one set vrestraint v\_/2:5/phi,PSI,xi1 1. 30. endfor build string "se ala his trp" set vrestraint v\_/2/phi,xi1,xi2 ,{-60.,-60.,120.} 0.5, 45. name="bb" set vrestraint v\_/2/phi,xi1,xi2 ,{ 60.,-60.,120.} 0.5, 45. name="cc" montecarlo v\_/2/phi,xi1,xi2

Note that in the command a special PSI torsion specification is used for traditional residue attribution.

### set values of internal coordinates

```
set vs_[ add ] { r_value | R_arrayOfValues }
```

sets specified variables to a given value(s) (for angles the value must be in degrees). If rarray  $R_arrayOfValues$  is specified, its values are assigned sequentially to the variables. It the array is shorter

than the selection, the values are applied periodically. Option add means increment by the specified value rather than set to this value.

Examples:

Note that in the second command a special PSI torsion specification is used for traditional residue attribution.

# set positional variables to place a molecule to polyhedral vertices

set vs\_grid i\_vertex i\_NofVertices

(order of arguments is important!) sets specified 2 variables ( normally a virtual planar angle and torsion angle ) to the values such as to put a molecule in the vertices of tetrahedron (i\_NofVertices=4), octahedron (6), cube (8), icosahedron (12) or dodecahedron (20). Used to sample uniformly the surface of globular molecules. Values of i\_NofVertices other than above are not allowed. The polyhedron is built around the origin. The size of the polyhedron is determined by v\_//bvtl variable which is a virtual bond length from the origin to the first virtual atom (vtl) of the two attached to each molecule. To check how polyhedrons are generated look at this example:

```
read object "complex"
display virtual a_//ca,c,n | a_//vt* only
color molecule
set a_1//vt1
                    # set vt1 of a_1 to its center of mass
set a_2//vtl
                    # set vt1 of a_2 to its center of mass
set a_2//vt1  # set vt1 of a_2 to its center of mass
set v_1//bvt1 0.1 # move a_1 to the origin (0.1 to avoid a singularity)
set v_2//bvt1 30. # offset a_2
                     # this is for a_2 to hop around a_1
for i=1, 20
  set v_2//avt1,fvt1 grid i 20
endfor
                     # this is for a_2 to rotate need the same location on a_1
for i=1,12
  for j=1,3
    set v_//avt2,tvt3 grid i 12
    set v_//tvt2 j*120.
  endfor
endfor
```

# set size and position of ICM graphics window

```
set window [ i_xLeft i_yDown ] i_xSize i_ySize [ margin= r_ ...]
```

sets the position and/or size (only size if 2 arguments are given) of the graphics window. Four arguments are in pixels. If you need to display in a fixed size window from a script we recommend to use the set window command first and then the display command.

In the off-screen mode (see the display off command) set window is accompanied by recentering of the molecular image with margin=  $r_{-}$  ... and other center options.

Example:

```
# square 700x700 window in the upper left corner
set window 570 30 700 700
display window
set window 300 300
write image window=3*View(window) # hi-res. image
```

# 2.20.64. show

show information about specified ICM-shell objects in your shell-window. Show is similar to the list command, but it gives you more information, covers a broader range of subjects and allows the user to show constants, subsets and expressions. However, in contrast to the list command, show does not understand wildcards.

Option full will show arrays and shell variables which are grouped into tables (the components of tables are hidden by default). The same option full temporarily sets l\_showSpecialChar to yes when sarrays are shown.

### show site

```
show site [ ms_] [ seq_1 seq_2.. ]
```

show sites assigned to the selected molecules ms\_ or sequences. By default all the sites of the current object are shown.

### show shell variable

show arg1 arg2 ...

```
show ICM-shell variable, constant, subsets, or expressions. One needs to separate arguments by comma only if two consequtive arguments are numbers, and the second on is a negative number constant.
```

Examples:

```
show azurins[3:20] # show a fragment of the alignment
show a b a*b # two arrays and their product
show Sin({1.3.5.}) # another array
show 2., -3. # without the comma, it will show -1.
show m_crn # map (m_crn) header information and
# the map sections
```

# show key

show key

show commands bound to key-strokes. Allowed keys: F1, ... F12, Ctrl-F1, ... Ctrl-F12, Ctrl-A, ... Ctrl-Z, Alt-A, ... Alt-Z. See also the set key command.

#### show map

show { map | mapName }

show the current or the specified map in text format. Example:

```
buildpep "AKSD"
make map potential Box(a_) "ge"
display m_ge {1 2 3 0 4 5 6}
show m ge
m_ge> written in ZYX mode (z-sections). Symmetry group #0
  Box {sect0,row0,col0, sect,row,col} = {-30,-8,-21, 32,16,28}
  Cell {A,B,C, angles(deg)} = {14.000,8.000,16.000, 90.00,90.00,90.00}
  Nof intervals (at x, y, z) = \{28, 16, 32\}
  Min/max/mean/rms density = -20.000000, 20.000000, -0.182712, 12.082560
:***######*:...::::*****####*
:***##**::...::::::********
:****
::***....
::***:....
--- 13 / 32 ---
           # shows pages
```

#### show objects, molecules, residues, atoms and variables

show { *os*\_|*ms*\_|*rs*\_|*as*\_|*vs*\_ }

show selected atom(s) as\_, residue(s) rs\_, molecule(s) ms\_, object(s) os\_, or variable(s) vs\_, respectively.

Examples:

show a\_\*. # all objects show a\_\*.\* # all molecules of all objects show a\_2.\* # all molecules of the second object show a\_\* # all molecules of the current object show a\_/ala # all alanines of the current object show a\_1//c\* # carbons of the 1st molecule of the current object show v\_2.a//phi,psi

```
Data fields for objects :
```

show object
# a\_objectName. type n\_Mol n\_Res n\_waters resolution object\_name
1 a\_def. Type: ICM Mol: 1 Res: 4 def <\*\*\* the current object
2 a\_1dna. Type: X-Ray Mol: 3 Res: 532 Wat: 216 Resol: 2.20 thymidylate synt..</pre>

These fields can be accessed with the following functions:

- object name: Name ( os\_ )
- object type: Type( os\_, 2) # returns "X-Ray","NMR","ICM",etc.
- number of molecules: Nof(ms\_), e.g. Nof( a\_2.\* )
- number of residues: Nof( rs\_), e.g. Nof( a\_2.\*/\* )
- resolution: Resolution( os\_), e.g. Resolution( a\_2. )
- number of waters: Nof ( *water\_selection* ), e.g. Nof ( a\_2.w\* )
- full name: Namex( os\_), e.g. Namex( a\_2. )

Data fields for **molecules** :

read pdb show a_*	"la36"		
Name	n_residue:	s first_res_name	object_name
i Mole	ecule N_Rea	S	Object
1 a	544	ile	1a36
2 b	22	dpa	1a36
3 C	22	dpa	1a36
4 wl	1	hoh	1a36
5 w2	1	hoh	1a36

These and other molecule attributes can be accessed with the following functions:

- mol. name: Name( ms\_)
- mol. type: Type( ms\_, 2) # field not shown Returns. "Nucl", "Amino", "Hetatm" etc.
- number of residues: Nof( rs\_), e.g. Nof( a\_2.\*/\* )

### show alias

show aliases

show all currently defined aliases. To show a specific alias, use the

alias aliasName

command (e.g. alias cd).

# show alignment

```
show alignments [ color ]
```

show currently loaded alignments. Option color color sesidues in the alignment by type.

### show area

show area { surface | skin } [ mute ] [ as\_1 [ as\_12 ] ]

Calculates the area of the solvent-accessible surface or molecular surface (so called skin), respectively. You can specify for which atoms you want to calculate the surface (selection *as\_1*).

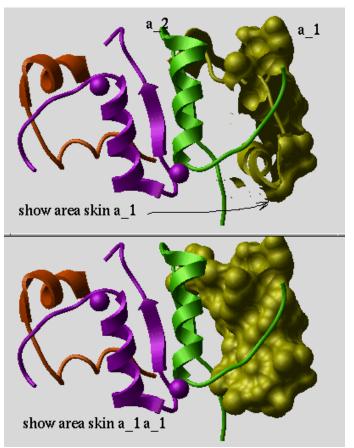
You can also additionally specify the environment for these selected atoms, i.e. the neighbors which you want to take into account in the surface calculation.

The two most popular modes are the following:

- measuring the surface area of some atoms being a part of the whole system (e,g, a\_1 a\_\* or just a\_1, the top picture)
- measuring the surface area of a group of atoms as if they are the only atoms that exist in space (e.g. a\_1 a\_1 the bottom picture).

In essence, two optional selections [ $as_1$  [ $as_12$ ]] impose a mask on atom pairs, so that only pairs in two selections are considered. If only the first selection is specified, the second one is assumed to be *all atoms*. The two *reasonable* choices for the second selection are *all atoms* (the default), and the *repetition* of the first selection (acts as if not other atoms are present in the system). In all cases, the second selection must include the atoms of the first one, e.g.

show area skin a\_1 a\_1,2



The area will be stored in r\_out and the number of triangles used in the "skin" construction in i\_out.

### show atoms

show as\_

shows properties of the selected atoms. Example:

```
build string "se ala"
show surface area
show a_//c*
```

2.20.64. show

```
      Atom Res
      Mol Obj
      X
      Y
      Z
      Occ
      B
      MMFF Code
      Xi Chrg formal Grad Area Grp as_

      ca
      1
      ala
      al def
      -2.748
      0.000
      -2.245
      1.00
      20.0
      1
      113
      C
      1
      0.06
      0
      0.0
      0.5
      _
      a.d

      cb
      1
      ala
      al def
      -2.329
      -1.202
      -3.093
      1.00
      20.0
      1
      113
      C
      0
      0.0
      7.3
      _
      a.d

      c
      1
      ala
      al def
      -4.247
      -0.000
      -1.935
      1.00
      20.0
      3
      121
      C
      0
      0.45
      0
      0.0
      34.2
      c
      a.d
```

The fields:

Field

Description

- ···
atom name
residue number+[symbol] and name
molecule name
object name
coordinates
occupancy (from 0. to 1.)
B-factor (positive)
MMFF atom code
ICM atom code
chirality number $(0,1,2,3)$
partial charge
formal charge
solvent accessible surface area
electrostatic group (atoms can not be separated)
selection expression

# show atom type

show atom type : \*show \*atom \*type \*mmff [ { s\_pattern | i\_type } ]

shows atom types stored in the icm.cod file. The mmff option allows to check the Merck Force Field atom type.

### Examples:

show atom type # show all ICM types ----- atom codes ------# # hb hd sf na icd VW wt # 0 0 0.000 0.00 ? atcd 0 0 atcd 1 1 1 0 1.008 0.00 h atcd 2 3 1 0 1.008 0.00 h . . . show atom type mmff "\*cation\*" # cations show atom type mmff "\*iron\*ion\*" # do we have iron ions? show atom type mmff "?C=\*" # what types are connected to doubly-bonded carbon ?

```
show atom type mmff "[!C]*ring*"
    # non-carbon types in rings
show atom type mmff 32
    # some oxygens
-------
Symb.Typ.[V] Description {formal charge}
O2CM 32 [1] oxygen in carboxylate anion
OXN 32 [1] N-oxide oxygen
O2N 32 [1] nitro oxygen
O2NO 32 [1] nitro-group oxygen in nitrate
...
```

# show clash

```
show clash [ as_1 [ as_2 ] ] [ r_vwReductionRatio ]
```

shows all the interatomic distances between two atom selections which are shorter than the sum of van der Waals radii multiplied by the *r\_vwReductionRatio* parameter (0.8 by default). IMPORTANT: this will work only for the ICM–objects. Use the show energy "vw" command (and pay attention to the current fixation) to precalculate interaction lists. The output will show the actual distance and the ratio of this distance and the sum of radii. Mark the two atoms of interest, separated by a logical OR, and paste it into another command if necessary.

```
See also: display clash, undisplay clash. Visualize the strained atoms with show a_//G or display a_//G.
```

Example:

build string "se ala his trp glu"
randomize v\_//\*
display
show clash a\_//c\* a\_//c\* # clashes between carbons
show clash a\_//c\* a\_//c\* 0.7 # more tolerant test
display clash

# show color list

show color [mute]

shows list of colors defined in the file icm.clr and stores the output list in the S\_out string array. Option mute suppresses output to the screen but still saves to the S\_out array (useful for scripts)

See also: color command.

An example:

shov	v color		
		colors	
1	black		#000000
2	white		#ffffff
3	grey		#878787
4	blue		#0065ff
5	red		#ff0000

. . .

Example of show color mute use in a script:

```
if (Exist(view)) then # check if graphics is active
    show color mute # saves a list of colors in S_out
    for i = 1, Nof(S_out)
        color background $S_out[i]
        pause
    endfor
endif
```

# show arrays as parallel vertical columns

```
show column array1 array2 .... [s_fileName][separator=s_Separators][comment=s_Comment]
```

shows several arrays in a multi-column format.

See also: write column, show database, write database.

#### Example:

```
resnam = {"ala" "glu" "arg"}
reschg = { 0., -1., 1. }
show column resnam reschg
show column separator=":" comment="Example table" resnam reschg
```

# show comp\_matrix

show comp\_matrix

shows residue comparison matrix used by the alignment algorithms.

See also: set comp\_matrix, read comp\_matrix.

### show table in database format

show database { table | array1 array2 .... }

shows several arrays or a table in a database format.

See also: read database show column, write database.

Example:

```
resnam = {"ala" "glu" "arg"}
reschg = { 0., -1., 1.}
show database resnam reschg
```

# show drestraint

show drestraint [ as\_select [ as\_select ]] [ center ] [ mute ] [ r\_violation ]

shows distance restraints. Arguments:

- optional *as\_select* atom selection arguments specify atom pairs to be considered. Attention, the as\_out selection can not be used as an argument since it is redefined by the command.
- *r\_violation* : if the *r\_violation* distance is specified, only the restraints deviating from the upper or lower bounds by *r\_violation* are shown.
- center : If center option is specified the violation is measured with respect to the target value of the distance restraint and optionally only the distances greater than *r\_violation* are reported.
- mute option: allows one to fill out the as\_out selection and calculate the number of selected drestraints (i\_out) without actually reporting them. It is useful for scripts.

Output:

- as\_out atomic selection of all atoms for which the specified criteria have been satisfied
- i\_out reports the **number** of selected drestraints

See also: drestraint and drestraint type.

# show drestraint type

show drestraint types

shows available drestraint types as defined in the icm.rst file. The numbered global or local types can be used to impose distance restraints. The other types are fixed and are used to impose disulfide bonds or peptide bonds.

### show energy

show energy [ mute ] [ s\_termString ] [ vs\_ ] [ as\_select1 [ as\_select2 ] ]

calculates and shows values of currently set or explicitly defined in *s\_termString* energy terms (e.g. " $\forall w$ , el"). If *vs\_* selection is specified, only the selected variables will be unfixed. The initial fixation will be restored after completion. Two additional atom selections may specify a subset of atom pairs that should be considered by the minimization procedure. Note that the contribution from the "14" energy term is not displayed separately. It is included in the " $\forall w$ " contribution. If you want to display it separately, use the more straightforward Energy("14") function.

**Important:** the boundary element electrostatics is the most computationally heavy term. It is activated if electrostatic term el is switched on and preference electroMethod is set to "boundary element". The most demanding part is the calculation of the boundary and its characteristics. Therefore, for multiple calculations with the same boundary we recommend to use make boundary and delete boundary commands.

# show energy gradient

show gradient

show gradient calculated by the minimize or show energy commands.

# show hbond

show hbond [ r\_maxHbondDistance ]

calculates and outputs the list of hydrogen bonds. The real argument *r\_maxHbondDistance* defines the upper bound of the distance between a hydrogen and a potential hydrogen acceptor to place the pair to the hydrogen bond list. Default value of *r\_maxHbondDistance* parameter is 2.5 A. Number of identified hydrogen bonds is saved in i\_out. To display/undisplay hydrogen bonds, use display hbond and undisplay hbond commands. Hydrogen bonds can also be calculated by the minimize and show energy commands provided that the hydrogen bond term is switched on.)

# show bond exact

show hbond exact

calculate the hydrogen bonding energy according to the distributed electron density geometry. Used in virtual screening to evaluate a score.

# show table in html format

show html T\_[link T\_.S\_1 s\_linktype1 T\_.S\_2 s\_linktype2 ...]

show the  $T_{\text{table}}$  with HTML tags. Interpret web links according to the web link types described in the WEBLINK.DB array.

See also:

- write html *s\_file T\_* [ link ... ] write the html document to a file
- web  $T_{[link...]}$  directly show the table in the web browser.

Option none suppresses the table title and the copyright notice.

### Example:

show html SR link SR.NA2 "PDB"

### show iarray

show iarrays

show integer arrays defined in the shell. It shows names, dimensions and the first elements of arrays. The I\_out array contains the output of some functions and commands and is always in the shell.

ii={1 2 3 4 5 6 76}

2.20.64. show

iii=Count(10)
show iarray
----- iarrays ----I\_out[1:1] { 0, ... }
ii[1:7] { 1, ... }
iii[1:7] { 1, ... }

# show integers

show integers

show all integer shell variables. Example:

```
show integer
----- integers -----
                111
 а
 autoSavePeriod
               10
 defSymGroup
                1
                 0
 i_out
 minTetherWindow 20
 mnRemarks
                 3
 mnSolutions
                50
 . . .
```

# show label

show labels

show graphics string labels to find out their number. Then the labels can be addressed as label 1, label 2 etc.

See also: display string\_label

# show library

show libraries

show loaded ICM-libraries. It's a lot of stuff, enter 'q' to exit.

# show link

show link

show links between sequences, alignments and corresponding molecules of 3D objects.

# show logical

show logicals

shows all logical shell variables in ICM-shell. Example:

```
aa=yes
```

```
show logical
----- logicals -----
aa yes
l_alignProfiles yes
l_antiAlias yes
l_antiAliasGLfix no
l_autoLink yes
l_bpmc yes
...
```

# show mol

show mol *as\_select* 

shows selected atoms in the mol file format. See also: read mol and write mol.

# show mol2

show mol2 as\_select

shows selected atoms in the mol2 –file format (file extension .ml2). See also: read mol2 "file" and write mol2 "file".

# show molecule

show molecules

shows all molecules of all objects currently in icm-shell. This command is identical to show a\_\*.\*

# show object

show objects

shows all molecular objects currently in icm-shell. This command is identical to show a\_\*.

The same result is achieved with the list  $a_*$ . command.

# show pdb

show pdb as\_select

show selected atoms in the PDB file format.

See also: read pdb "file", and write pdb "file".

# show preferences

show preference

shows all icm preference variables in icm-shell (e.g. .

```
show preferences
...
atomSingleStyle = "tetrahedron"
    1 = "tetrahedron" # current choice
    2 = "cross"
    3 = "dot"
...
```

# show profile, rarray, real, sarray, string

show profile | rarray | real | sarray | string

shows all objects of specified type(s) in icm-shell. E.g. E.g.

show sarray rarray

#### show residue

show residues

shows all residues in all molecules of all molecular objects. This command is equivalent to

show a\_\*.\*/\*

### show residue type

show residue types

show names and characteristics of compounds described in the icm.res and user ICM residue libraries.

### show segment

show segment [ ms\_ ]

show segment representation of 3D structure of a protein for the selected molecules *ms*\_(all molecules of the current object by default).

See also assign sstructure segment, ribbonStyle, display ribbon.

#### show sequence

show sequences [number][{ fasta | swiss | pir | gcg | msf }]

show all sequences or the specified sequence  $seq_i$  in one of specified formats. The default format is the fasta format. Option number defines if the residue numbers are added.

Three logicals: 1\_showSstructure, 1\_showSites, and 1\_showAccessibility control the display of a corresponding additional information aligned with the sequence.

Example:

```
read sequence "GTPA_HUMAN"
show sequence swiss GTPA_HUMAN
read pdb "llbd"
show surface area
make sequence
Info> sequence 1lbd_m extracted
show llbd_m # you see relative accessibilities in 0-9 scale
l_showAccessibility = no
show llbd_m
```

# show stack

show stack [ [ i\_FromConf ] i\_ToConf ]

show the following parameters of the conformations currently residing in the conformational stack.

- iconf a slot number
- ener total energy as calculated before the conformation was stored
- rmsd the distance (either Cartesian or angular RMSD) between the current conformation of the object and the stack conformation calculated according to the compare command.
- naft the number of visits AFTER the last improvement of energy
- nvis the total number of visits to this slot; since new conformation are only compared with the last stack conformation the conformations may drift and cover a large area than described by the vicinity parameter

### show table

show *T\_table* [database]

shows the specified table. See also: show  $html T_{-}$ . Database index tables are exceptions, show T\_index will show all the entries of the related database. To see members of an index table type the index table name and press TAB.

### show terms

show terms[all]

shows the active energy/penalty terms. With option all it shows all the terms available. The result is saved in the s\_out string. See also: set terms, delete terms.

# show tethers

show tethers [ mute ] [ as\_select ] [ r\_minDeviation ]

Shows tethered atoms with deviation larger than  $r_minDeviation$  (0. by default) and returns these atoms in as\_out. Option mute is used when you just want to get a selection (as\_out) of strongly deviated atoms.

See also: display tethers.

# show version

show version

show characteristics of the current ICM executable. Part of this string containing the version number is returned by the Version() function.

### show vrestraints

show vrestraint [ vs\_ ]

shows vrestraints imposed on the internal variables of ICM molecular object.

# show vrestraint type

show vrestraint types

shows types of vrestraints. These types are loaded from the icm.rst file.

# show volume

show volume { surface | skin } [ mute ] [ as\_1 [ as\_2 ] ]

Calculates the volume confined by the solvent-accessible surface or molecular surface (so called "skin"), respectively. Two optional selections [ $as_1$  [ $as_2$  ]] impose a mask on atom pairs, only those to be considered. If only the first selection is specified, the second one is assumed to be all atoms. The volume will be stored in r\_out and the number of triangles used in the skin construction in i\_out.

Examples:

# calculate volume of blobs of map density.

show volume [ map ] [ I\_indexBox[1:6] ] [ r\_Threshold ]

Contour electron density map at a given *r\_Threshold* and calculate the volume of the high–density blobs. Defaults:

- take the current map;
- contour the whole map;
- use threshold value from the ICM-shell real variable mapSigmaLevel.

Threshold is expressed in the units of standard deviations from the mean map value, i.e. 1. stands one sigma over the mean. The volume will be stored in  $r_out$ . See also: make grob  $m_$ .

Examples:

```
read map "crn" # load m_crn map
show volume m_crn 3. # calculate volume inside the
```

# 2.20.65. sort

a family of sort commands (sort objects, molecules in ojbect, array/arrays or sort tables by their columns ).

# sort array(s)

```
sort [ reverse ] [ number ] sort_key_array [ array2 array3 ... ]
```

sort one or several integer, real or string arrays. The first array is the sort key. By default ordering is lexicographic for string arrays and by increasing arithmetic value for integer and real arrays.

Options:

- reverse: reverse the sense of comparisons.
- number: enforce sorting according to arithmetic value for string arrays.

Examples:

```
a={3 2 1 5 7 4 6}
b=Sin(a*50.)
c={"three" "two" "one" "Five" "Seven" "four" "Six" }
show column a b c
show column a b c
sort reverse b a c
show column a b c
sort c b a
show column a b c
```

# sort molecular objects by mass or a user field

sort object [ field = i\_Field ]

resorts **all** molecular objects by the specified user field (see the set field command, and the Field function). If the field is not specified, the objects are sorted by their mass.

# sort molecules in an object by mass or a user field

```
sort os_ObjectSelection [ field = i_Field ]
```

resorts the molecules in each of the selected non-ICM objects by the specified user field (see the set field command, and the Field function). If the field is not specified, the molecules are sorted by molecular mass. An ICM object can be stripped, resorted and then converted again.

#### sort table

sort [reverse] [number] table.keyArray1 [reverse] table.keyarray1 [reverse] ...

this command sorts all the arrays of the table so that all the listed *table.keyArrays* are applied sequentially with descending priority. Each array can be followed by the reverse option to change the sorting order.

Examples:

sort stack

sort conformations in a stack according to their energies. New energies can be assigned to the same conformations with the set stack energy command.

### 2.20.66. split

can split grobs or tables into individual components.

#### split grob

split g\_complexGrob [ s\_rootGrobsName] [ i\_maxNofGrobs] [ r\_minGrobSize ]

divide disconnected parts of a graphics object into a bunch of separate graphics object sorted according to their size (measured as the number of vertices). The maximal number of new grobs is defined either by  $i\_maxNofGrobs$  explicitly or by the MnGrobs parameter. The latter can be redefined in the icm.cfg configuration file. The  $i\_maxNofGrobs$  option allows to retain only larger pieces. Grobs will be sorted according to their number of points and named by adding their sequential number to the input grob name or  $s\_rootGrobsName$ , if specified. Splitting allows you to treat, display, and measure disconnected parts separately. The split command is invaluable for protein cavity analysis.

You can also limit the **size** of the objects generated by the command by providing the real argument with the minimal number of vertices you want in a grob.

See also: Volume(g\_), Area(g\_), Xyz(g\_).

Example:

```
read object "crn"
make grob skin a_//cb a_//cb
split g_crn
display grob smooth # display as one smooth surface
```

```
split [t_tableName]
```

split table into individual arrays.

Example:

group table t  $\{1\ 2\ 3\}$  "a"  $\{2\ 3\ 4\}$  "b" # t.a t.b arrays split t # a and b arrays

### Splitting an object into separate molecules

split object

There is no such command, but if you want to split a molecular object into separate molecules, you can simply copy the object and delete unwanted molecules in each copy. Example:

```
copy a_ "b"
delete a_b.!1 # delete all but the first molecule
write a_b. "b" # contains only the first molecule
#
copy a_ "c"
delete a_c.!2 # delete all but the second molecule
write a_c. "c" # contains only the second molecule
#
#etc..
```

### 2.20.67. sprintf

sprintf [ append ] s\_formatString arg1 arg1 arg2 arg3 ... [ name= s\_outputStringName]

Print to the s\_out string, or the s\_outputStringName specified after the name= option. The same syntax as printf command, but the result is not displayed.

Example in which string outStr is the destination:

sprintf "mncalls = %d\n",mncalls name="outStr"

## 2.20.68. store

```
store conf [ i_slotNumber ] [ r_energy ] [ number = i_nOfVisits ]
```

store current conformation into specified slot of the conformational stack. By default it puts the conformation into the first free slot, or appends it to the end. The energy, by default, is automatically extracted from the previous energy evaluation, or taken from  $r_{energy}$  if explicitly provided. The total number of visits (nvi) is set to 1 by default, or Example:

store torsion type [ vs\_ ]

store temporary torsion types for new ICMFF potential for a given molecule (not used in the commercial distribution).

## 2.20.69. ssearch

is a systematic search through torsion space combined with local minimization.

- you may globally optimize any set of energy/penalty terms including electrostatics, solvation, entropy, density correlation etc.
- you may search an arbitrary subset of variables
- you may allow full local minimization after each systematic change
- you may search only through centers of the preferred local multidimensional zones (for example rotamers) which is more efficient than an even grid sampling
- you may perform both the global search (the full [-180.,180.] range) and the local search (grid search around the current conformation).

ssearch[local][vs\_Ssearch[vs\_minimize]][as\_select1[as\_select2]]

systematically changes *vs\_Ssearch* variables and carries out energy minimization with respect to the *vs\_minimize* variables after each systematic conformational change. The lowest energy conformation is loaded from the conformational stack at the end of the procedure. By default every variable from *vs\_Ssearch* selection goes through nSsearchStep evenly distributed values. The step therefore is 360 deg. over *nSsearchStep*. Option local imposes the grid locally around the current values of *vs\_Ssearch* variables. In this case the program uses ssearchStep parameter.

If you want to prevent the procedure from automatically writing the stack of best conformations to a file set the autoSavePeriod variable to zero.

See also montecarlo.

Example:

2.20.68. store

read object "crn"  $\ \$  good old crambin ssearch v\_/14/x\*  $\ \$   $\ \$  place optimally Asn14 side-chain

# 2.20.70. strip

strip os\_object

strip an ICM–molecular object from its ICM attributes and reduce it into a pdb–object. The latter are still good for graphics, superposition, basic geometric manipulations etc. Also, some chemical operations, e.g. attaching chemical groups are best performed on simpler pdb–objects. Stripping may save you a lot of memory as well.

String is also used to perform operations which are not allowed for ICM object, but are allowed for simpler PDB objects (for example dragging individual atoms with a mouse)

These commands include:

- deleting hydrogens
- make bond auto

### 2.20.71. superimpose

```
superimpose[[align|ali_][exact]]as_selectStatic as_selectMovable
```

superimpose the second object onto the first one using selected atoms or residues as equivalent points. Selections may by of any level: atom selection  $as\_$  residue selection  $rs\_$  molecular selection  $ms\_$  or object selection  $os\_$ . The option defines how the two sets are aligned (the residue alignment may be explicitly provided as the *ali\_* argument, and the objects are linked with the alignment):

#### alignment options:

- Default (no options): Residue alignment: by residue number. Atom alignment: by atom name for pairs of identical residues or pairs of close residues (F with Y; B with D,N; D with N; E with Qor Z, Q with Z), for other residue pairs only the backbone atoms ca,c,n,o,hn,ha are aligned.
- align option: *Residue* correspondence is established by *sequence alignment* using the ICM ZEGA alignment Abagyan, Batalov, 1997 *Atom* alignment: by atom name (see the default option).
- exact option: Residue matching is ignored. Two atom selections are directly sequentially aligned. Numbers of atoms in two selections must coincide.
- align exact option: Residue alignment: Needleman and Wunsch. Inside residue atoms are aligned sequentially and regardless of the name.

Number of equivalent atom pairs is saved in i\_out; resulting RMSD is saved in r\_out; a selection of atoms in the "static" object used for superposition is saved in as\_out, that of "movable" object in as2\_out.

Note that if the movable object is of ICM-type it is preferable to have all six virtual variables unfixed ( e.g. unfix  $V_movableObj.//?vt*$ ). Otherwise, if some or all of them ( $V_//?vt*$ ) are fixed, you will get a warning, and only the partial minimization of the RMS distance possible with the given degrees of freedom will be performed.

See also: Rmsd(), Srmsd().

### 2.20.72. then

is one of the ICM flow control statements, used to perform conditional statements.

See also if, elseif, and endif.

### 2.20.73. transform

transform molecular objects to symmetry related positions.

transform {ms\_|g\_grob} R\_12transformationVector

transform molecules (ms\_) or graphics objects according to the transformation vector.

See also these two examples: (example 1 and example 2).

You can also manually move molecules with respect to each other on the graphics screen by using the connect  $ms_{-}$  command to choose the molecules which can be moved separately.

transform[copy]ms\_[i\_number][s\_symgroup] {box |as\_selection r\_radius}

transform molecules  $ms\_$  according to the specified transformation. *i\_number* is a symmetry operation number in an array of all operators of a space group. The symmetry transformations are defined in a 12\*n real array where each chunk of 12 real values defines 3x3 rotation matrix and translation vector {a4,a8,a12}. The complete 4x4 transformation matrix looks like this:

a1	a2	a3	a4
a5	a6	a7	a8
a9	a10	a11	a12
0.	0.	0.	1.

If *i\_number* exceeds the number of space group symmetry transformations the symmetrical images in up to 26 surrounding cells are created. This operation is only possible, if symmetry information (sym.group name and cell dimensions) is defined for the object. Usually PDB and CSD files contain the above information, it is preserved upon conversion. Use the Cell() or the Symgroup() functions to find out if the space group is defined. If not, you may assign it to the object with the set symmetry object command. In a special case of *i\_number=0*, the object is placed in the "primary" subunit of the cell (e.g. in sym.group "P 21 21 21" that is 0 < x < a, 0 < y < b, 0 < z < c/4; currently, the *i\_number=0* option is supported only for groups 1 and 19).

Example:

read mol2 "ex\_mol2" # several small molecules
display a\_4.
build hydrogen a\_4. # added and displayed

## 2.20.74. translate

translate { os\_ | ms\_ | g\_grob } [ add ] [ symmetry ] R\_3translationVector

translate the center of mass of the specified object(s) ( $os_{-}$ ) or molecule(s) ( $ms_{-}$ ) to a specified position, or by a  $R_{-}3translationVector$  vector if option add is specified. You can also move molecules/objects interactively with the mouse after the connect command.

symmetry option With the symmetry option the R\_3translationVector should be in **fractional** coordinates. Option add translates by the specified vector from the current position. Without add the program tries to identify a compensating shift to a position in which the center of gravity of the selected molecule(s) has minimal positive fractional coordinates.

Examples:

translate a\_2  $\{2. 0. 1.\}$  # shift second molecule by the vector

## 2.20.75. undisplay

Opposite to display. To get rid of the whole graphics window for fast calculations use

undisplay window

Examples of the undisplay command:

```
undisplay ribbon
                          # ribbon display not needed any more
undisplay g_icos
                          # a graphics object not needed any more
# who cares about water molecules ...
undisplay string
                          # see also "delete label" command
undisplay a_//h*
                          # who cares about hydrogens ...
undisplay hbond a_1./1:29
                          # ... and, hence, about H-bonds
undisplay tether a_/12:20
undisplay box
undisplay cursor
undisplay origin
                          # undisplay the coordinate frame
undisplay window
                          # delete GL graphics window
```

## 2.20.76. unfix

unfix [ only ] Vs\_select

unfix (set free) specified variables (such as bond lengths, angles and phases or torsions) in an ICM-object. Opposite to fix command. This operation can be applied to the current object only (use set object os\_newObj first).

**Important:** since it only makes sense to unfix variables which are currently fixed, use Vs\_select which selects among ALL (both free and fixed) variables, as opposed to vs\_ which selects only from FREE variables.

Examples:

```
# only this loop has free torsions now
unfix only V_/8:18/phi,PSI,H,M,P
```

Note that PSI torsion references is used for traditional residue attribution

### 2.20.77. unix

unix unix\_shell\_command

issues a UNIX shell command from ICM. The ICM process waits until the UNIX shell process has completed. "unix" must be the first word in the command. **Important**: Construction

if ( <condition> ) unix unix\_command

is illegal. Use

```
if ( <condition> ) then
    unix unix_command
    endif
```

instead.

To pass the ICM-shell variables to the *unix\_shell\_command* use integer, real or string ICM-shell variables, protected with dollar sign (\$) prefix. **Important:** passing ICM-shell variables to the UNIX command is impossible if you use an alias name (e.g. ux) instead of the original unix command.

Examples:

### 2.20.78. wait

wait for the child ICM processes to finish, quit the child processes

wait

allows to synchronize multiple ICM processes spawned by the fork command.

- for the parent process: wait until all the child processes spawned with the fork command are finished.
- for the child processes: quit the spawned ICM process

See also: iProc, nProc.

### 2.20.79. web

web Args ...

a family of web commands.

```
web[delete]u_EntrezRequest
```

invoke a web browser call to the WWW page of the Entrez server at NCBI and display your database entries with links. The format of the requests is described at http://www3.ncbi.nlm.nih.gov/Entrez/linking.html. The s\_webEntrezLink string defines the location of the link and the webEntrezOption preference defines the type of the Entrez report. The delete option will temporarily set l\_confirm=no for this command.

Examples:

```
web sp ||GTPA_HUMAN  # Swissprot by ID
web sp |P07497|  # Swissprot by accession
web emb |U01234|  # EMBL by accession
web pdb |crn|  # PDB by ID (a single chain entry)
web pdb |2ins|a  # PDB by ID and chain character
web 1111111  # NCBI nonredundant database
```

### 2.20.80. web table: shows an icm table with a web browser

```
web[delete][s_file]T_[link T_.S_1 s_linktype1 T_.S_2 s_linktype2 ...]
```

ICM invokes a web browser call. If you do not have a browser (defined by the s\_webViewer string) currently running under your login, the call will fail. The command presents the  $T_{table}$  in your web browser window. Optional web links are interpreted according to the web link types described in the WEBLINK.DB array.

See also:

• `writehtml{write html [s\_file] T\_ [ link ... ]} - write the html document to a file • `showhtml{show html T\_ [ link ... ] }

- show the table in the icm text window.

Example:

```
show html SR link SR.NA2 "PDB"
#
read sequence "GTPA_HUMAN.seq"
find profile
show SITES
web SITES
web SITES link SITES.AC "AUTO"
```

# 2.20.81. while

while

```
is one of the ICM flow control statements, used to perform a loop in the ICM-shell calculations.
```

See also: for, endwhile.

### 2.20.82. write

write stuff to a disk file. Logical variable l\_confirm defines if you'll be prompted whether to overwrite an existing file with the same name. Use option delete to delete (or overwrite) the existing file unconditionally.

For the list of ICM-objects you can write, and formats you can choose, see read and show commands. Generic syntax:

```
write[append|delete]{variable|constant|expression} s_fileNameRoot[.ext]
```

See also corresponding read commands.

### write alignment

```
write[alignment][msf] ali_Name[s_fileName]
```

write alignment *ali\_Name* to a file. Default extension is .ali. Note: if alignment is only a group of unaligned sequences, generated by the group command, the result will be just a multiple sequence file, rather than an alignment file (there will be no dashes at the end).

The default ICM format for an alignment looks like this:

#>ali sh3 # Consensus Fyn Spec Eps8 #Fyn	#.^.YD%+~#~# K~#~##.~~~WW.#. ~~.~ VTLFVALYDYEARTEDDLSFHKGEKFQILNSSEGDWWEARSLTTGET DETGKELVLALYDYQEKSPREVTMKKGDILTLLNSTNKDWWKVEVNDRQ KTQPKKYAKSKYDFVARNSSELSM-KDDVLELILDDRRQWWKVRNSGD EEEEEEEEEEEE
# Consensus Fyn Spec Eps8 #Fyn	G%#P##. GYIPSNYVAPVDSIQ GFVPAAYVKKLD GFVPNNILDIMRTPE EEEGGGGEEE
# nID 7 Lmin	61 ID 11.5 %

The lines starting from hash (#) are comments and are not required

The length of each alignment block is controlled by the sequenceLine parameter (default value is 60). If you want to save a long alignment as one unwrapped block, increase this value (e.g. sequenceLine=1000)

### Writing sequences in the alignment order

The sequences can be written in the alignment order with the following commands (they can be store in a little macro)

```
macro wrSeqAli ali_ s_file ("seq.fasta")
    l_showSstructure = no
    seqname = Name(ali_) # Name returns sarray of sequence names
    for i=1,Nof(seqname)
        write sequence fasta append $seqname[i] s_file
    endfor
    endmacro
```

#### Resorting alignment in the order of sequence input.

Upon alignment the source sequences get reordered according to similarity. If you want to keep the original order you may use the reorderAlignmentSeq macro described in the Align(*ali\_I\_newOrder*) section and then write an alignment:

```
read sequence s_icmhome+"zincFing"
group sequence aaa
align aaa
reorderAlignmentSeq aaa
write ali_new  # reordered alignment
```

See also: String( *ali\_*) function.

#### write iarray

write[iarray]I\_name[s\_fileName]

#### write rarray

write [ rarray ] R\_name [ s\_fileName ]

#### write sarray

write [ sarray ] S\_name [ s\_fileName ]

#### write matrix

write [ matrix ] M\_name [ s\_fileName ]

write an array or a matrix to a disk file. Default file extensions are .iar, .rar, .sar, or .mat, respectively.

See also: read iarray, read rarray, read sarray, read matrix.

#### write several arrays

write[{ column | database ] } array1 array2 .... [ s\_fileName ]

write arrays in the column or database format to a disk file. Default file extension is .db

See also: read database.

#### writing tethers

If you imposed tethers between you current object and another object and you want to quit the session and then restore you setup, you can use the following trick:

# first let us create an object a\_ly6. tethered to template a\_x. read alignment s\_icmhome+"sx" read pdb s\_icmhome+"x" build model ly6 a\_x.m # a new object a\_ly6. created and tethered # write string String( a\_//T) "tTz.str" # tethered model atoms write string String( a\_//Z) "xTz.str" # x-template atoms write object a\_x,ly6. "tx.ob" # quit # % icm read object "tx.ob" read string "tTz.str" name="tTz" read string "xTz.str" name="tTz" set tether \$xTz \$tTz exact # tethers restored

#### write table

write T\_table1 [ T\_table2 .. ] [ field= s\_delimiter ] [ s\_fileName ]

write the  $T_{table}$  table to a disk file \*.tab. It will have two header lines with table name and field name information, followed by the values.

The default extension .tab is appended automatically. The ICM text table format has a header which allows to read this table back to icm with the read table command

Example:

```
group table t {1 2 3} "a" {"one","two","three"} "b"
t1=t[2:3]
write t t1 "tt" # write both tables in one file
delete table # read both tables
```

```
write T_table1 [ header ] [ separator= s_delimiter ] [ s_fileName ]
```

if the separator or the s\_fieldDelimiter variable contain just a simple symbol (e.g. comma or tab), ICM will write a comma-separated or tab-separated table with the first line containing the field names, e.g.

```
group table t {1 2 3} "a" {"one","two","three"} "b"
write t header separator="," "t.csv"
unix cat t.csv
"a","b"
"1","one"
"2","two"
"3","three"
write t separator="," "t.csv" # without header
unix cat t.csv
```

"1","one" "2","two" "3","three"

To read a table in comma-separated format (without headers), use the following commands:

```
s_fieldDelimiter="," # or "\t"
read column group name="t" "t.csv"
```

See also write database T and write column.

#### write column

```
write column array1 array2 .... [s_fileName] [separator= s_Separators]
```

write arrays in a multi-column format to a disk file.

Examples:

read column s\_icmhome + "res.tab" # amino acid properties
write column aa flexInd "tm.tab" # two columns

If you want to write all the entries of an ICM-table you may do the following.

Examples:

```
read column s_icmhome + "res.tab" # a set of isolated arrays
group table RES $s_out # create an ICM-table RES (s_out : array names)
write RES # write in the 'table' layout
write database RES # write table RES in the 'database' layout
```

Default file extension is .col.

See also: read column, show column. read table, show table.

#### write database

write database [ html ] { array1 array2 .... | table } [ s\_fileNameRoot ]

write several arrays or a table in a database format to a file (usually tables are written in a multi column format). This command can also be used to save a subset of arrays of a table in a specific order. Option html writes the table with appropriate HTML tags. See also read database write table, show database.

Example:

```
resnam = {"ala" "glu" "arg"}
reschg = { 0., -1., 1.}
write database resnam reschg "a" # default extension ".db" will be added
#
group table t resnam reschg
write database t.reschg t.resnam "a" # reverse the order</tt>
```

### write drestraint

write drestraint [ as\_ ] [ s\_fileNameRoot ]

write distance restraints of the current object to a file.

See also: drestraints and drestraint types.

### write drestraint type

write drestraint types

write drestraint types to a file. You may define your own types with the set drestraint type command or by editing a \*.cnt file.

### write factor

write [ factor ] factor\_Name [ s\_factorFileNameRoot ]

writes crystallographic structure factors to a file.

### write grob

```
write[grob][g_name][s_fileName][append]
```

write/append a graphics object (grob) to a disk file. If  $g_name$  is not specified, all grobs are written. Default file extension is .obj

See also: write image, write postscript.

#### write html

write html s\_htmlFileName T\_ [ link T\_.S\_1 s\_linktype1 T\_.S\_2 s\_linktype2 ... ]

writes the  $T_{\text{table}}$  with HTML tags to a file. Interpret web links according to the web link types described in the WEBLINK. DB array.

See also:

show html T\_ [ link ... ] – show the html formatted table in the text window
web T\_ [ link ... ] – directly show the table in the web browser.

Example:

write html SR "results.htm" link SR.NA2 "SP" # link to Swissprot

#### write image

```
write image [{ png | targa | gif | rgb }] [ display ] [ print ] [ postscript [{ print |
preview }]] [ compress ] [ stereo ] [{ color | bw }] [ window= I_xyPixelSizes ] [ s_fileName ]
```

2.20.82. write

write the current screen image to a file. The default image file format is tif . The png-format is the most compact and is recommended for web-publishing. The default settings are stored in the IMAGE table. Some of them can be overridden by the following options:

- display allows to view the saved image or postscript image file. The viewer is defined by the s\_imageViewer variable for targa, gif, rgb and tif images and by the s\_psViewer variable for the postscript images.
- postscript write Adobe postscript–bitmap file rather than TIFF–file. See also write postscript command which generates **vectorized** scalable high quality postscript files.
- preview add low-resolution preview to postscript file for some EPS-compliant image viewers (i.e. Irix showcase®). Resolution, and therefore the size, of the added preview is defined by the IMAGE.previewResolution (default 10).
- print print the postscript file. It will not work for non-postscript images, in which case you may use the display option and print from your image viewing program instead.
- compress use packbits lossless compression standard for **.tif** files. Compression of this kind is currently a standard feature of all baseline TIFF–reading programs. Compression is a standard feature of the .gif and .png formats.
- stereo generate stereo image even from the mono display. Tiff-files preserve the image screen dimensions for each image in a stereo-pair. Stereo-base for postscript files is controlled by the IMAGE.stereoBase parameter and equals 2.35" (60mm) by default.
- color or bw color or black-and-white options surpass IMAGE.color logical variable.
- window= *I\_xyPixelSizes* generate image of any arbitrarily large resolution (e.g. window=3\*View(window) to triple the resolution). Suppose that you want to make a poster of 4613 by 2888 pixels. This resolution is not achievable on a 1200x1024 screen. The image area will be divided into many squares and the program will merge them into one image of large resolution. This option will not work with string labels.

Example:

```
nice "lcrn" # resize the image
delete label
IMAGE.compress = no #just a plain uncompressed image
write image window={4000,2700} # for slides
write image window=2*View(window) # double the res.
```

IMAGE.generateAlpha logical variable controls if the alpha channel information is added to the SGI rgb and tif image files. This additional channel describes opacity of the image pixels and makes the background transparent. Images generated with alpha channel can be nicely superimposed in the IRIX showcase since their backgrounds are transparent.

Examples:

```
display a__1crn. ribbon
```

See also: write grob, write image, write postscript.

#### write index

write index [ swiss | mol | mol2 | fasta ] T\_dbDescription s\_outFile

calculate and write index for a database described by the control table  $T_dbDescription$ . This table contains information about the database file (files) and fields to be indexed. It may have the following components in the header:

- DIR string directory name
- FI sarray of database files
- EXT extension of the database files

After the header there is a string array containing the list of fields. To create this table either define it in a file or use the group table command. All text fields (except data) are hashed for fast searching. The fasta option allows to index the NCBI nonredundant databases. See also: makeIndexChemDb macro to do indexing in one step, mol, mol2.

#### Example:

```
group table t {"ID","DE","KW","SQ"} "fd" header "/data/swissprot/" \
    "DIR" {"sprot"} "FI" ".dat" "EXT"
    # we created control table t
write index swiss t "/data/icm/inx/SWISS.inx" # make index and save to a file
read index "/data/icm/inx/SWISS.inx" # read index
show SWISS[2:5]
show SWISS.ID=={"12AH_CLOS4","1431_LYCES","B3AT_CHICK"}
read sequence SWISS.DE=="DNA-BINDING"
```

#### write index blast

write index sequence s\_blastRootFileName

create a set of blast-formatted binary files for searches with the find database command. The command will use all the sequences currently loaded into the ICM-shell and will create the following compact binary files (the first three files are the same as those generated by the setdb blast command):

- *name.bsq* binary sequences
- name.atb pointers
- name.ahd sequence headers
- name.bsa relative solvent accessibilities for each residue. This information

If you want to do the opposite (i.e. given the three or four blast files, generate one fasta sequence file), use the

find database write *s\_DBpath* output= *s\_fastaFile* 

command.

Simple example (indexing can also be done with the blast setdb routine):

```
read sequence "aaa.seq" # fasta formatted
write index sequence "./aaa"
delete sequences
a=Sequence("SFDGHASGDFSHGASFDHAGS")
find database a "./dom" 0.001
```

An example in which the sequences+accessibilities are generated from objects:

```
read sarray "domain.list" name="dom" # e.g. {"lcrn","labc.a/l2:l15/",..}
for i=1,Nof(dom)
read object s_icmhome+"data/xpdb/"+dom[i]
show area surface mute # calculate the absolute acc. areas
make sequence a_1  # the relative areas are calculated and assigned to seq.
delete a_*.
endfor
write index sequence "./dom" # files .bsq .atb .ahd .bsa created
delete sequences
```

#### write library

write library [append] [auto] as\_entryAtom [exit= as\_exitAtom] s\_libFileRoot

save a selected molecule, residue or a fragment as an ICM-library entry. Use set charge, set bond type and, possibly, build hydrogens before writing an entry. We recommend you to do this operation in an interactive session: display your molecule and Ctrl-Click the first and last atoms if needed. There are two different situations:

- 1. the molecule/residue/fragment does not belong to an ICM-type object. For example, you have a pdb-file with a new molecule you would like to create an ICM-library entry from. In this case do NOT use option auto and note that the resulting entry will only be a draft, since energy parameters of atoms (atom codes plus related types of van der Waals, hydrogen bondings solvation), as well as parameters of torsions, bond angles, phase angles, and bond lengths will have to be further manually adjusted. Enter the command and you will be prompted for the first and the last atoms of the entry. The purpose of this procedure is to create a regular ICM-tree, create extra bonds if there are cycles and give atoms unique names. Some additional editing of the entry may be required to correct fixed and free torsions suggested by the program. To declare a certain variable free, enter '+' in the appropriate field.
- 2. the molecule/residue/fragment belongs to an ICM-type object. In this case you may use option auto since all the information is there already. The program only needs to extract the molecular subtree according to the specified selection.

Example:

```
BS glu # build glutamic acid residue
strip # convert it to a non-ICM object
write library a_def.a1/1/hg2 "./tm" name ="new" auto # reroot it
```

```
# Now the entry atom is a_//hg2
build string "se new" library ="./tm" # read the rerooted residue
display
```

#### write map

write m\_map [ s\_fileName ]

write specified map to a binary file with specfied file.

write { map | m\_map1 m\_map2 ... }

write all maps or specified maps to corresponding files ( the names for the files are generated from map names, the m\_ prefix is removed from the file names).

```
write xplor m_map ... [ s_fileName ]
```

write the specified map to a Xplot-formatted file.

Example:

```
make map "ge,gc" potential Box(a_)
m_gc... done
Info> Map m_gc created. GridStep=0.50 Dimensions: 16 11 17, Size=2992
m_ge... done
Info> Map m_ge created. GridStep=0.50 Dimensions: 16 11 17, Size=2992
write m_ge m_gc
Info> 1 map written to file ge.map
Info> 1 map written to file gc.map
```

#### write model: update or create the loop database file

write model [append] s\_lpsFile

writes a compressed representation of the protein structure to the specified loop file ( "def.lps" by default ). To create a large database, read the object list and write a loop over all objects, e.g.

```
# prepare pdbUniq list and ..
read sarray "pdbUniq.li"
for i=1,Nof(pdbUniq)
   read object s_xpdbDir+pdbUniq[i]
# add further filters
   write model append "icm.lps"
   delete object
endfor
```

To make the program use this file, redefine the LIBRARY.lps file name to, say "./icm.lps"

#### write mol

```
write mol [ exact ] as_select [ s_fileName ]
```

write selected atoms in the mol -file format. By default the formal charges (see the set charge command) are saved. Options

- exact: preserve the ICM-atom names (like c1, c2).
- charge: write the MCHG section containing the atomic real charges.

See also read mol "file", show mol "file".

#### write mol2

write mol2[exact][formal] as\_select[s\_fileName]

write selected atoms in the mol2-file format (extension .ml2). Options:

- exact preserves the ICM-atom names (like c1, c2).
- formal writes formal atomic charges instead of the real ones. Adds USER\_CHARGES (XXXXXX) tag to the header

See also read mol2 "file", show mol2 "file".

#### write object

write object [ options ] [ as\_selection] [ s\_fileName [ rename ] ]

write an ICM molecular object (or many selected ICM–objects) in binary ICM format to a file. A single object can be renamed in the file according to the *s\_fileName*, if option rename is specified. **Important**: only **whole** ICM object may be written by this command, and file extension will always be . ob.

Options (defaults shown in bold):

- append : append to a multiple–object file
- rename : rename the single object to *s\_fileName* (leave out path and extension).
- strip: write a stripped object (i.e. drop information about variables and rigid bodies present in an object of the ICM type).
- auto={yes | no}: if yes the program automatically identifies which atom requisites to save. For example, if molecule is displayed, the view will be saved with the object. Properties such as occupancy and charge are considered essential if the values are not identical for all the atoms.

If auto=no, the OBJECT table controls the output.

- occupancy={yes | no} : occupancy field
- charge ={ yes|no} : partial atomic charges
- bfactor ={ yes|no} : b-factors
- display = {yes| no} : the current view of your molecular object(s), including graphics planes The written display attributes are automatically restored upon reading of the object.
- library={yes|**no**} : currently not used.

See also: read object, write pdb, OBJECT, strip.

Example:

```
read object "crn"
build string "se ala his" name="AH" # second object named "AH"
write object a_2. "alahis" rename # rename obj. to "alahis"
display a_1./1:40 ribbon # display and save with graphics attributes
display a_1./12 cpk
display a_2. xstick
write object a_*. "twoobj" display=yes # both objects in one file
write object a_1. append "twoobj" # yet another object
```

### write object simple

```
write object simple [ as_selection] [ s_fileName ]
```

write a compressed object. The information preserved in the compressed description of the object is limited to 3 coordinates and certain atom names (non-protein atom names will not be preserved and reduced to just one character) plus all residue and molecule requisites. For a PDB-type file, a simple object is the most compact for store and fastest to read. They are used in the compact fold library.

#### write pdb

write pdb[exact][charge][nosort][as\_selection][s\_fileName]

write a molecular (sub)object in PDB format. Normally atoms of each amino acid are sorted in the following order:

ATOM	19	Ν	GLN	0	3	-4.565	0.000	-4.592	1.00	20.00
ATOM	20	CA	GLN	0	3	-4.712	0.000	-6.037	1.00	20.00
ATOM	21	С	GLN	0	3	-6.194	0.000	-6.420	1.00	20.00
ATOM	22	0	GLN	0	3	-7.063	0.000	-5.549	1.00	20.00
<i>the</i>	rest<	/i>								

Also the n-terminal nitrogen and its hydrogens are assigned to the first amino acid. Options are the following:

- charge saves atomic charges instead of occupancies and atomic radii instead of B-factors;
- exact keeps the names of hydrogen atoms the same as in ICM objects (i.e. the first character is 'h'). Without this option names of hydrogen atoms are transformed like this:

```
h11 ==> 1H1
h12 ==> 2H1
```

• nosort retain the original ICM order of atoms

Default file extension is .pdb.

See also: write object, read pdb.

#### write postscript

```
write postscript[display][stereo][preview][{ color | bw | dash]}[i_quality][
r_gammaCorrection][s_filename]
```

create **vectorized** postscript model of the screen image. Instead of the bitmap snapshot this command generates lines, solid triangles and text strings corresponding to the displayed objects. Since the postscript language is directly interpreted by high-end printers, the printed image may be even higher quality than the displayed image. The final resolution is limited only by the printer since the original image is not pixelized. Warning: there may be inevitable side-effects for some types of solid images at the intersection lines of solid surfaces (i.e. large scale cpk-representation, hint: use display skin instead).

The default settings are stored in the IMAGE table. Some of them can be overridden by the following options and arguments:

- reverse makes white background in the saved postscript file.
- display allows to view the saved postscript file. The viewer is defined by the s\_psViewer variable.
- stereo generate stereo image even from the mono display. Stereo-base is controlled by the IMAGE.stereoBase parameter and is 2.35" (6cm) by default.
- preview generates postscript preview according to the IMAGE.previewer command string and the IMAGE.previewResolution parameter.
- color or bw color or black-and-white options surpass IMAGE.color logical variable.
- dash is a great variant of the black–and–white option to generate lines of different width and style. The line colors of your screen image are interpreted according to the following table:
- gold double solid black line
- pink triple solid black line
- magenta dash1
- orange dash2
- brown dotted line
- the rest solid black line

#### Examples:

```
display a__crn. # display wire model of crambin
color a_//ca,c,n pink  # triple width backbone
color a_/arg/!ca,c,n magenta # dashed lys side chains
# zoom your picture to fill the whole graphics window
write postscript dash stereo display
```

\*  $i_quality$  (default=3, possible range: 1:100) – defines a parameter in a smoothing procedure. Each side of an elementary triangle is divided into  $i_quality$  sections and color of all the  $i_quality^2$  smaller triangles is calculated to yield smooth transitions. Optimal value of the parameter depends on an image. Only large scale images may require  $i_quality$  values above 10. Only in an extreme case of a single triangle on a page with red, blue and green vertexes, one may need  $i_quality$  of 100.

\* *r\_gammaCorrection* allows to lighten or darken the image by changing the *gamma* parameter. A gamma value that is greater than 1.0 will lighten printed picture, while a gamma value that is less that 1.0 will darken it. You may adjust your gamma correction parameter for your printer with respect to your display and add this setting to the \_startup file.

Examples:

See also: write image, write grob.

### write pov

write pov [image ] [r\_aspectRatio] [s\_fileName]

writes a pov-ray object file which can be processed with the pov-ray ray-tracing program.

Example:

```
buildpdp "ala his trp"
display cpk
make grob image
write pov "x"
% pov-ray x.pov
```

#### write segment

write segment [ append ] [ s\_fileName ]

writes a simplified description of protein topology generated by the assign sstructure segment command to a file. You can append your description to the provided foldbank.seg file.

Examples:

```
read object "crn"
assign sstructure segment a_*
write append segment "myseg"
```

See also: find segment, read segment.

#### write sequence

```
write { sequence | seq_ } [ { fasta | swiss | pir | gcg | msf } ] [ s_fileName ]
```

write all sequences or the specified sequence  $seq_{to}$  to a file in one of specified formats. The default format is the fasta format.

#### write session

write [ s\_fileName ]

write commands from an ICM session to a file. Default file name is "\_session.icm". This is a simple text file with icm commands. Feel free to edit the file

Example:

```
..
a=1
history 10
write session
Info> 4 history lines written to file _session.icm
```

See also: history command.

### write stack

write stack [ s\_fileName ]

write the current state of the conformational stack to a disk file. Default file extension is .cnf.

See also: show stack, delete stack, read stack, read conf.

#### write vs\_var

write [ vs\_variables][ s\_fileName ]

write a variable selection vs\_ to a disk file.

Default file extension is .var.

See also: read variable.

# 2.21. Functions

ICM-shell functions are an important part of the ICM-shell environment. They have the following general format: FunctionName (arg1, arg2, ...) and return an ICM-shell object of one of the following types: integer, real, string, logical, iarray, rarray, sarray, matrix, sequence, profile. alignments, maps, graphics objects (grob) and selections.

The order of the function arguments is fixed in contrast to that of commands. The same function may perform different operations and return ICM-shell constants of different type depending on the arguments types and order. ICM-shell objects returned by functions have no names, they may be parts of algebraic expressions and should be formally considered as 'constants'. Individual 'constants' or expressions can be assigned to a named variable. Function names always start with a capital letter. Example:

show Mean(Random(1.,3.,10))

### 2.21.1. Abs

absolute value function.

Abs (real) - returns real absolute value.

Abs (*integer*) – returns integer absolute value.

Abs (*rarray*) – returns rarray of absolute values.

Abs (*iarray*) – returns iarray of absolute values.

Examples:

# 2.21.2. Acc

accessibility selection function. It returns residues or atoms with relative solvent accessible area greater than certain threshold. **Important:** The surface area must be calculated before this function call. The Acc function just uses surface values, it does not reevaluate them. Therefore, make sure that the show area command (or show energy, minimize, etc. with the "sf" surface term turned on), has been executed before you use the Acc function. If you specify the threshold explicitly, it must range from 0.0 to 1.0, otherwise it is set to 0.25 for residue selections and 0.1 for atom selections.

Acc ( rs\_, [ r\_Threshold ] )

- returns residue selection, containing a subset of specified residues  $rs_{-}$  for which the ratio of their current accessible surface to the standard exposed surface is greater than the specified or default threshold (0.25 by default). ICM stores the table of standard residue accessibilities in an unfolded state calculated in the extended Gly-X-Gly dipeptide for all amino acid residue types. It can be displayed by the show residue type command, or by calling function Area(*s\_residueName*), and the numbers may be modified in the icm.res file.

The actual solvent accessible surface, calculated by a fast dot–surface algorithm, is divided by the standard one and the residue gets selected if it is greater than the specified or default threshold. ( $r_Threshold$  parameter is 0.25 by default).

Acc (as\_select, [r\_Threshold])

- returns atom selection, containing atoms with accessible surface divided by the total surface of the atomic sphere in a standard covalent environment greater than the specified or default threshold (0.1). Accessibility at this level does not make as much sense as at the residue level. The standard surface of the atom was determined for standard amino-acid residues. Note that hydrogens were **NOT** considered in this calculation. Therefore, to assign surface areas to the atoms use

```
show surface area a_//!h* a_//!h*
command or the
show energy "sf"
command.
```

You may later propagate the accessible atomic layer by applying Sphere( $as_{,1,1}$ ), where 1.1 is larger than a typical X-H distance but smaller than the distance between two heavy atoms. (the optimal *r\_Threshold* at the atomic level used as the default is 0.1, note that it is different from the previous ).

### Examples:

```
# let us select interface residues
read object s_icmhome+"complex"
                                  # display all surface residues
show surface area
display Acc( a_/* )
                                  # now let us show the interface residues
display a_1,2
color a_1 yellow
color a 2 blue
                                  # calculate surface of
show surface area a_1 a_1
                                  # the first molecule only
                                  # select interface residues
                                  # of the first molecule
color red Sphere(a_2/* a_1/* 4.) Acc(a_1/*)
read object "crn"
show energy "sf"
display
display cpk Acc(a_//* 0.1) # display accessible atoms
show surface area
                             # prior to invoking Acc function
                             # surface area should be calculated
color Acc(a_/*) red
                             # color residues with relative
                             # accessibility > 25% red
```

### 2.21.3. Acos

arccosine trigonometric function Returns angles in degrees.

Acos (real | integer) – returns the real arccosine of its real or integer argument.

Acos (rarray) – returns the rarray of arccosines of rarray elements.

#### Examples:

### 2.21.4. Acosh

inverse hyperbolic cosine function.

```
Acosh (real | integer ) - returns the real inverse hyperbolic cosine of its real or integer argument.
```

Acosh (rarray) - returns the rarray of inverse hyperbolic cosines of rarray elements.

#### Examples:

```
print Acosh(1.) # returns 0
```

print Acosh(1) # the same
print Acosh({1., 10., 100.}) # returns {0., 2.993223, 5.298292}

# 2.21.5. Align

aligns two sequences with the Needleman and Wunsch algorithm with zero gap end penalties (ZEGA). The ZEGA-statistics of structural significance (Abagyan, Batalov, 1997) is given and can be additionally evaluated with the Probability function. The reported pP value is -Log(Probability,10).

Align ([sequence1, sequence2 [{ area | distance | superimpose } [i\_window] [r\_seq\_weight]]] ) - returns ZEGA- alignment. If no arguments are given, the function aligns the first two sequences in the sequence list.

Returned variables:

- i\_out the number of identical residues in the alingment
- r\_out contains Log ( *Probability\_of\_structural\_dissimilarity* ) only for pairwise alignments
- r\_2out percent identity of the alignment.

### Simple pairwise sequence alignment

Align()

Align (*seq1 seq2*) – returns an alignment. The alignMethod preference allows you to perform two types of pairwise sequence alignments: "ZEGA" and "H-align". If you skip the arguments, the first two sequence are aligned.

Example:

### Aligning with custom residue weights or weights according to surface accessible area

Align(seql seq2 area)

Option area will use relative residue accessibilities to weight the residue-residue substitution values in the course of the alignment (see also accFunction).

The weights must be positive and less than 2.37. Try to be around or less than 1. since relative accessibilities are always in [0,1.] range. Values larger than 2.37 do not work well anyway with the existing alignment matrices and gap parameters. Use the Trim function to adjust the values, e.g. Trim(myweights, 0.1,2.3)).

E.g.

read pdb "11bd"

2.21.5. Align

This can also be used to assign *custom* weights with the following commands

```
set area seq1 R_weights \mbox{ \# must be } > 0. and less than 2.37 Align( seq1 seq2 area )
```

#### Introducing positional terms into the alignment score.

Align (seq1 seq2 M\_positionalScores)

If sequence similarity is in the "twilight zone" and the alignment is not obvious, the regular comp\_matrix{residue substitution matrix} is not sufficient to produce a correct alignment and additional help is needed. This help may come in a form of the positional information, e.g. histidine 55 in the first sequence must align with histidine 36 in the second sequence, or the predicted alpha–helix in the first sequence preferably aligns with alpha–helix in the second one.

In this case you can prepare a matrix of extra scores for each pair of positions in two sequences, e.g.

```
seq1 = Sequence("WEARSLTTGETGYIPSA")
seq2 = Sequence("WKVEVNDRQGFVPAAY")
Align()
# Consensus W.#. .~~.~G%#P^
seq1 WEARSLTTGETGYIPS--
seq2 WKVE--VNDRQGFVPAAY
m = Matrix(17,16,0.)
m[10,4] = 3. # reward alignment of E in seq1[10] and E in seq2[4]
Align(seq1 seq2 m)
# Consensus W.# E ~G%#P^
seq1 WEARSLTTGE----TGYIPS--
seq2 WKV-----EVNDRQGFVPAAY
```

The alignSS macro shows a more elaborate example in which extra scores are prepared to encourage alignments of the same secondary structure elements.

Warning. The alignment procedure is very subtle. Avoid values comparable with gap opening penalty.

#### Local structural alignment

Two types of structural alignments or mixed sequence/structural alignments can be performed with the Align function.

Align ( $seq_1 seq_2 distance [i_window] [r_seq_weight]$ ) – performs local structural alignment, using **distance RMSD** as structural fitness criterion. The RMSD is calculated in a window  $i_window$  and the dynamic programming algorithm combines structural scores with sequence alignment scores if  $r_seq_weight>0$ .

Align ( $seq_1 seq_2 superimpose [i_window] [r_seq_weight]$ ) – performs local structural alignment, using **superposition followed by coordinate RMSD** calculation as structural fitness criterion. The RMSD is calculated in a window  $i_window$  and the dynamic programming algorithm combines structural scores with sequence alignment scores if  $r_seq_weight>0$ .,

In both cases the function uses the dynamic algorithm to find the alignment of the locally structurally similar backbone conformations.

The alignment based on optimal *structural superposition* of two 3D structures may be different from purely sequence alignment

Preconditions:

- sequences must be linked to 3D molecules to access the coordinate information;
- two 3D structures must have a superimposable subsets

The residue-label-carrying atoms (see the set label a\_command) will be used for structural superpositions.  $r\_seq\_weight$  is used to add sequence aminoacid substitution values to the 3D similarity signal.

### Extracting pairwise alignment sequences from a multiple alignment

Align (ali\_, seq\_1, seq\_2) – returns a pairwise sub-alignment of the input alignment *ali*\_, reorders of sequences in the alignment according to the order of arguments.

### Extracting a multiple alignment of a subset of sequences from a multiple alignment

Align (ali\_, *I\_seqNumbers*) – returns a reordered and/or partial alignment. Sequences are taken in the order specified in *I\_seqNumbers*.

Examples:

### Resorting alignment in the order of sequence input with the Align (ali\_, I\_seqNumbers) function.

Load the following macro and apply it to your alignment. Example:

```
macro reorderAlignmentSeq( ali_ )
nn=Name(ali_)  # names in the alignment order
ii=Iarray(Nof(nn))
j=0
for i=1,Nof(sequence)  # the original order
ipos = Index( nn, Name(sequence[i] ) )
if ipos >0 then
    j=j+1
    ii[j] = ipos
```

```
endif
endfor
ali_new = Align( ali_ ii )
keep ali_new
endmacro
```

#### Deriving an alignment from tethers between two 3D objects

Align  $(ms_)$  – returns alignment between sequences of the specified molecule and the template molecule to which it is tethered. The alignment is deduced from the tethers imposed.

This function may be used to save the alignment after interactive editing.

Example:

```
build string "se ala his leu gly trp ala" "a" # obj. a
build string "se his val gly trp gly ala" "b" # obj. b
set tether a_2./1:3 a_1./2:4 align # impose tethers
show Align(a_2.1) # derive alignment from tethers
write Align(a_2.1) "aa" # save it to a file
```

## 2.21.6. Angle

calculates planar angle in degrees. Returns real value.

Angle ( $as\_atom$ ) – returns the planar angle defined by the specified atom and two previous atoms in the ICM-tree. For example, Angle( $a\_/5/c$ ) is defined by C–Ca–N atoms of the 5–th residue. You may type:

print Angle( # and then click the atom of interest.

Angle (*as\_atom1*, *as\_atom2*, *as\_atom3*) – returns the planar angle defined by three atoms.

Angle (*R\_3point1*, *R\_3point2*, *R\_3point3*) – returns the planar angle defined by the three points.

Angle (*R\_vector1*, *R\_vector2*) – returns the planar angle between the two vectors.

Examples:

### 2.21.7. Area

calculates surface area.

Area (grob) - returns real surface area of a solid graphics object.

See also: the Volume( grob ) function, the split command and How to display and characterize protein cavities section.

Area (as\_) – returns rarray of pre-calculated solvent accessible areas for selected atoms as\_.

Area ( $rs_$ ) – returns rarray of pre–calculated solvent accessible areas for selected residues  $rs_$ . These accessibilities depend on conformation.

Area ( $rs_type$ ) – returns rarray of maximal standard solvent accessible areas for selected residues  $rs_$ . These accessibilities are calculated for each residue in standard extended conformation surrounded by Gly residues. Those accessibilities depend only on the sequence of the selected residues and do NOT depend on its conformation. To calculate normalized accessibilities, divide Area( $rs_$ ) by Area( $rs_$ type)

Example:

```
read object "lcrn"
show surface area
a=Area(a_/* )  # absolute conformation dependent residue accessilities
b=Area(a_/* type ) # maximal residue accessilities in the extended conformation
c = a/b  # relative (normalized) accessibilities
```

### 2.21.8. Area contact matrix

Area ( $rs_1 rs_2$ ) – returns rarray of **areas of contact** between selected residues. You can do it for intramolecular residue contacts, in which case both selections should be the same, i.e. Area( $a_1/* a_1/*$ ); or, alternatively, you can analyze intermolecular residue contacts, for example, Area( $a_1/A a_2/A$ ). See also the Cad function, and example in plot area in which a contact matrix is calculated via interatomic Ca–Ca distances. The table of the pairwise contact area differences is written to the s\_out string which can later be read into a proper table via: read column group name="aa" input=s\_out and sorted by the area (see below).

Example:

```
read object "crn" # good old crambin
s=String(Sequence(a_/A))
PLOT.rainbowStyle="blue/rainbow/red"
plot area Area(a_/A, a_/A) comment=s//s color={-50.,50.} \
link transparent={0., 2.} ds
read object "complex"
plot area Area(a_1/A, a_2/A) grid color={-50.,50.} \
link transparent={0., 2.} ds
```

Area (*string*) – returns the real value of solvent accessible area for the specified residue type in the standard "exposed" conformation.

**Important** : "pre-calculated" above means that before invoking this function, you should calculate the surface by show area surface, show area skin or show energy "sf" commands.

Examples:

build	<pre># build a molecule according to the sequence # from file def.se (default)</pre>
show area surface a = Area(a_//o*)	<pre># calculate surface area # individual accessibilities of oxygens</pre>
<pre>stdarea = Area("lys")</pre>	# standard accessibility of lysine

2.21.8. Area contact matrix

See also: Acc() function.

## 2.21.9. Asin

arcsine trigonometric function Returned values are in degrees.

Asin (*real* | integer)

- returns the real arcsine of its real or integer argument.

Asin(*rarray*)

- returns the rarray of arcsines of rarray elements.

Examples:

### 2.21.10. Asinh

inverse hyperbolic sine function.

Asinh(real)

- returns the real inverse hyperbolic sine of its real argument.

```
Asinh(rarray)
```

- returns the rarray of inverse hyperbolic sines of rarray elements.

Examples:

print Asinh(1) # the same
print Asinh({-1., 0., 1.}) # returns {-0.881374, 0., 0.881374}

## 2.21.11. Ask

interactive input function. Convenient in macros.

Ask ( s\_prompt, *i\_default* )

- returns entered integer or default.

Ask ( s\_prompt, r\_default )

- returns entered real or default.

Ask ( s\_prompt, l\_default )

- returns entered logical or default.

Ask ( s\_prompt, s\_default [simple] )

- returns entered string or default. Option simple suppressed interpretation of the input and makes quotation marks unnecessary.

Examples:

### 2.21.12. Atan

arctangent trigonometric function Returned values are in degrees.

Atan (real | integer)

- returns the real arctangent of its real or integer argument.

Atan (rarray)

- returns the rarray of arctangents of *rarray* elements.

Examples:

## 2.21.13. Atan2

arctangent trigonometric function. Returned values are in degrees.

Atan2(r\_x, r\_y)

- returns the real arctangent of  $r_y/r_x$  in the range -180. to 180. degrees using the signs of both arguments to determine the quadrant of the returned value.

Atan2 ( $R_x R_y$ )

- returns the rarray of arctangents of  $R_y/R_x$  elements as described above.

Examples:

## 2.21.14. Atanh

inverse hyperbolic tangent function.

Atanh (real)

- returns the real inverse hyperbolic tangent of its real argument.

Atanh (rarray)

- returns the rarray of inverse hyperbolic tangents of rarray elements.

Examples:

### 2.21.15. Atom

transforms the input selection to atomic level necessary since some of the commands/functions require specific level of selection.

```
Atom ( as_Obj_or_Mol_or_Res_selection )
```

- returns selection converted to the atomic level.

Atom (vs\_)

- returns atom selection (i.e. selection of atomic level) to which the selected variables vs\_ belong.

Examples:

See also: the Res, Mol, adn Obj functions.

## 2.21.16. Augment

creates augmented affine 4x4 space transformation matrix.

```
Augment (R_12transformationVector)
```

- rearranges the transformation vector into an augmented affine 4x4 space transformation matrix.

The augmented matrix can be presented as

a1	a2	a3	a4
a5	a6	a7	a8
a9	a10	a11	a12
0.	0.	0.	1.

where  $\{a1,a2,...a12\}$  is the *R\_12transformationVector*. This matrix is convenient to use because it combines rotation and translation. To find the inverse transformation simply inverse the matrix:

```
M_inv = Power(Augment(R_12direct),-1))
R_12inv = Vector(M_inv)
```

To convert a 4x4 matrix back to a 12-transformation vector, use the Vector(  $M_{4x4}$  ) function.

See also: Vector (the inverse function), symmetry transformations, and transformation vector.

Augment (R\_6Cell)

- returns 4x4 matrix of oblique transformation for given cell parameters { *a b c alpha beta gamma* }.

This matrix can be used to generate real coordinates from fractional coordinates. It also contains vectors A, B and C. See also an example.

Example:

```
display a__crn.  # load and display crambin: P21 group
obl = Augment(Cell( ))  # extract oblique matrix
A = obl[1:3,1]  # vectors A, B, C
B = obl[1:3,2]
C = obl[1:3,3]
g1=Grob("cell",Cell( ))  # first cell
g2=g1+ (-A)  # second cell
display g1 g2
```

Augment ( $R_3$ Vector) – appends 1. to a 3D vector {x,y,z} (resulting in {x,y,z,l}) to allow direct arithmetics with augmented 4x4 space transformation matrixes.

Augment ( $M_XYZblock$ ) – adds {1.,1.,...} column to the Nx3 matrix of with {x,y,z} coordinates to allow direct arithmetics with augmented 4x4 space transformation matrixes.

## 2.21.17. Axis

calculates rotation/screw axis corresponding to a transformation

```
Axis({ M_33Rot | R_12transformation })
```

- returns rarray with x, y, z components of the normalized rotation/screw axis vector. Additional information calculated and returned by the function:

- r\_out rotation angle (degrees);
- r\_2out helix rise;
- R\_out 3-rarray with a point on the axis.

See also: How to find and display rotation/screw transformation axis

### 2.21.18. Bfactor

crystallographic temperature factors or custom atom parameters.

Bfactor ( $[as_|rs_]$ [simple]) – returns rarray of b-factors for the specified selection of atoms or residues. If selection of residue level is given, the average residue b-factors are returned. B-factors can also be shown with the command show pdb.

Option simple returns a normalized b-factor. This option is possible for X-ray objects containing b-factor information. The read pdb command calculates the average B-factor for all non-water atoms. The normalized B-factor is calculated as  $(b-b_av)/b_av$ . This is preferable for coloring ribbons by B-factor since these numbers only depend on the ratios to the average. We recommend to use the following commands to color by b-factor:

```
color ribbon a_/ Trim(Bfactor( a_/ simple ),-0.5,3.)//-0.5//3. \# or color a_// Trim(Bfactor( a_// simple ),-0.5,3.)//-0.5//3. \# for atoms
```

This scheme will give you a full sense of how bad a particular part of the structure is.

See also: set bfactor.

Examples:

## 2.21.19. Boltzmann

returns the real Boltzmann constant = 0.001987 kcal/deg.

Example:

deltaE = Boltzmann\*temperature # energy

### 2.21.20. Box

the 3D graphics box function. This box can be displayed with the display box command or by left-double-clicking on a grob, and interactively moved and resized with the mouse. One can select atoms inside a box by this operation:  $as_Box()$ 

Box () – returns the 6-rarray with  $\{X_{min}, Y_{min}, Z_{min}, X_{max}, Y_{max}, Z_{max}\}$  parameters of the graphics box as defined on the screen.

Box (center) – returns the 6– rarray with {*Xcenter*, *Ycenter*, *Zcenter*, *Xsize*, *Ysize*, *Zsize*} parameters of the graphics box as defined on the screen.

Box (*as* [*r*\_*margin*]) – returns the 6- rarray with {*Xmin*, *Ymin*, *Zmin*, *Xmax*, *Ymax*, *Zmax*} parameters of the box surrounding the selected atoms. The boundaries are expanded by *r*\_*margin* (default: 0.0).

Examples:

```
build string "se ala his" ~\# a peptide display box Box(a_/2\ 1.2)~\# surround the a_/2 by a box with 1.2A margin color a_//* Box( )
```

Box ( { *g*\_|*m*\_ } [ *r\_margin* ] )

- returns the 6- rarray with {*Xmin,Ymin,Zmin,Xmax,Ymax,Zmax*} parameters of the box surrounding the selected grob or map. The boundaries are expanded by *r\_margin* (default: 0.0).

### 2.21.21. Bracket

bracket the grid potential map by value or by space.

Bracket ( m\_grid [ r\_vmin r\_vmax ] )

- returns the truncated map. The map will be truncated by value. The values beyond  $r_vmin$  and  $r_vmax$  will be set to  $r_vmin$  and  $r_vmax$  respectively.

Bracket (*m\_grid* [*R\_6box*])

- returns the modified map. All the values beyond the specified box will be set to zero. Example:

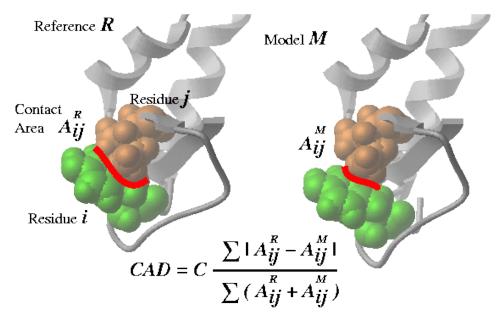
make map potential "gh,gc,gb,ge,gs" a\_1 Box()
m\_ge = Bracket(m\_ge, Box( a\_1/15:18,33:47 )) # redefine m\_ge

## 2.21.22. Cad

Contact Area Difference function to measure geometrical difference between two different conformations of the same molecule. Cad, as opposed to Rmsd, is contact based and can measure the difference in a wide range of model accuracies. Roughly speaking it measures the surface weighted fraction of native contacts. Can be used to evaluate the differences between several NMR models, the accuracy of models by homology and the accuracy of docking solutions.

Cad can measure the geometrical difference between two conformations in several different ways:

- between two conformations of the **same** protein based on full atom residue–residue contact area calculation, Cad(..)
- between two conformations of the same protein based on Cbeta-Cbeta distance evaluation (Cad(.. distance). ICM uses an empirically derived *ContactStrength*(*Cb-distance*) function.
- between two homologous structures based preservation of the residue contacts through the alignment ( Cad (.. alignment )). The contact strength in this case is also derived from the interresidue distances.



# Comparing two conformations of the same molecule via residue-residue contact conservation.

Cad(*rs\_A1*[*rs\_A2*]*rs\_B1*[*rs\_B2*][distance])

- returns the real contact area difference measure (described in Abagyan and Totrov, 1997) between two conformations A and B of the same set of residue pairs from two different objects. The set of residue pairs in each object (A or B) can be defined in two ways:

- by a single selection *rs\_A1* : all pairs between selected residues (is equivalent to *rs\_A1 rs\_A1* )
- by two residue selections *rs\_A1 rs\_A2*: cross pairs between two sets of selected residues (e.g. the contacts between two subunits)

The measure is a normalized sum of differences between residue–residue contact areas in two conformations. The measure was calibrated on a set of pairs of conformations. The average distortion due to a noncrystallographic symmetry is about 5%, the average CAD between a pair of models in an NMR entry is 15%. Note that the paper uses an additional factor of 1.8 (i.e. CAD=1.8\*Cad()) to bring the scale down to 0:100%, because about 40% of the contacts are trivial contacts between the neighboring residues. However, in evaluation of the docking solutions coefficient 1.8 should not be used. Loops are somewhat intermediate, but still a coefficient of 1.8 is recommended for consistency.

The whole matrix of contact area differences is returned in  $M_{out}$ . This matrix can be nicely plotted with the plot area  $M_{out}$  number ... command (see example). The full matrix can also be used to calculate the residue profile of the differences.

The table of the pairwise contact area differences is written to the s\_out string which can later be read into a proper table via: read column group name="aa" input=s\_out and sorted by the area (see below).

See also Area() function which calculates absolute residue-residue contact areas.

Options:

• distance option allows to compare approximations of the inter-residue contact areas by the Ca and Cbeta positions. This allows to calculated deformations between two homologous proteins which is not possible in the default mode in which two chemically identical molecules are compared. The residue pairs in two homologues are equivalenced according to the alignments linked to the molecules. Residues deleted in a homologue are considered to have zero contact.

```
# Ab initio structure prediction, Overall models by homology
  read pdb "cnf1" # one conformation of a protein
  read pdb "cnf2" # another conformation of the same protein
  show 1.8*Cad(a_1. a_2.) # CAD=0. - identical; =100. different
  show 1.8*Cad(a_1.1 a_2.1) # CAD between the 1st molecules (domains)
  show 1.8*Cad(a_1.1/2:10 a_2.1/2:10) # CAD in a window
  PLOT.rainbowStyle = 2
  plot area grid M_out comment=String(Sequence(a_1,2.1)) link display
# Loop prediction: 0% - identical; ~100% totally different
```

```
#
    CAD for loop 10:20 and its interactions with the environment
  show 1.8*Cad(a_1.1/10:20 a_1.1/* a_2.1/10:20 a_2.1/*)
#
     CAD for loop 10:20 itself
  show 1.8*Cad(a_1.1/10:20 a_1.1/10:20 a_2.1/10:20 a_2.1/10:20)
# Evaluation of docking solutions: 0% - identical; 100% totally different
  read pdb "expr"
                    # one conformation of a complex
  read pdb "pred"
                    # another conformation of the same complex
  show Cad(a_1.1 a_1.2 a_2.1 a_2.2) # CAD between two docking solutions
#
# ANOTHER EXAMPLE: the most changed contacts
  read object "crn"
  copy a_ "crn2"
  randomize v_ 5.
  Cad(a_1. a_2.)
  show s out
  read column group input= s_out name="cont"
  sort cont.1
  show cont
\# the table looks like this (the diffs can be both + and -):
#>T cont
#>-1-----3------3------
            a_crn.m/38 a_crn.m/1
  -39.
  -36.4
             a_crn.m/46 a_crn.m/4
  -32.1
             a_crn.m/46 a_crn.m/5
  -29.8
              a_crn.m/30 a_crn.m/9
  -25.2
              a_crn.m/37 a_crn.m/1
  42.5
              a_crn.m/43 a_crn.m/5
  45.1
              a_crn.m/44 a_crn.m/6
   45.2
              a_crn.m/43 a_crn.m/6
  55.3
              a_crn.m/46 a_crn.m/7
   56.
              a crn.m/45 a crn.m/7
```

Comparing two different, but structurally homologues proteins, via residue-residue contact conservation.

Cad(rs\_A1[rs\_A2]rs\_B1[rs\_B2]alignment)

# 2.21.23. Ceil

rounding function.

Ceil(r\_real[r\_base])

- returns the smallest real multiple of *r\_base* exceeding *r\_real*.

Ceil(R\_real[r\_base])

- returns the rarray of the smallest multiples of  $r_{base}$  exceeding components of the input array  $R_{real}$ . Default  $r_{base} = 1.0$ .

See also: Floor().

# 2.21.24. Cell

crystallographic cell function.

Cell({ *os*\_|*m\_map* })

- returns the rarray with 6 cell parameters {a,b,c,alpha,beta,gamma} which were assigned to the object or the map.

# 2.21.25. Charge

returns an rarray of partial electric charges of selected atoms, or total charges for residue, molecule or objects, depending on the selection level.

Charges can also be shown with a regular show as\_select command.

Charge({ *os*\_|*ms*\_|*rs*\_|*as*\_} [formal|mmff])

- returns rarray of elementary or total charges depending on the selection level.

- formal : return formal charges
- mmff : return *formal* charges calculated according to *mmff* atom types and rules. **Note:** do not confuse this option with a function to return the mmff charges.

Examples:

```
buildpep "ala his glu lys arg asp"
show Charge(a_1) # charge per molecule
show Charge(a_1/*) # charge per residue
show Charge (a_1//*) # charge per atom
                          # total electric charge of 15th residue
avC=Charge(a_/15)
avC=Sum(Charge(a_/15/*)) # another way to calculate it
show Charge(a_//o*)  # array of oxygen charges
# to return mmff charges:
set type mmff
set charge mmff
Charge( a_//* )
# to return total charges per molecular object:
read mol s icmhome+"ex mol.mol"
set type mmff
set charge mmff
Charge( a_*. )
```

See also: set charge.

# 2.21.26. Cluster

Cluster (I\_bitmasks i\_totalNofNeighbors i\_minNofCommonNeighbors)

Cluster (*M\_distances r\_maxDistance*)

function returns iarray of cluster numbers. This function identifies the *i\_totalNofNeighbors* nearest neighbors for each point and assembles points sharing the specified number of *common* neighbors in clusters.

Distance between two bitstrings is calculated using the Tanimoto algorithm.

All singlets (a single item not in any cluster) are placed in a special cluster number 0. Other items are assigned to a cluster starting from 1.

Example with a distance matrix:

```
# let us make a distance matrix D
# we will cook it from 5 vectors {0. 0. 0.}
m=Matrix(5,3) # initialize 5 vectors
m[2,1:3]={1. 0. 0.} # v2
m[3,1:3]={1. 1. 0.} # v3
m[4,1:3]={1. 1. 1.} # v4
m[5,1:3]={1. 0.1 0.1} # v5 close to v2
D = Distance( m ) # 5x5 distance matrix created
Cluster( D , 0.2 ) # v2 and v5 are assigned to cluster 1
Cluster( D , 0.1 ) # radius too small. All items are singlets
Cluster( D , 2. ) # radius too large. All items are in cluster 1
```

#### 2.21.27. Color

returns RGB numbers.

```
Color (g\_grob) – returns matrix of RGB numbers for each vertex of the g\_grob (dimensions: Nof (g\_grob),3).
```

See also: color g\_ M\_

Example:

```
Color (background)
```

- returns rarray of three RGB components of the background color.

# 2.21.28. Consensus

Consensus ( *ali\_* )

- returns the string consensus of alignment  $ali_{-}$ . The consensus characters are these: # hydrophobic; + RK; - DE; ^ ASGS; % FYW; ~ polar. In the selections by consensus a letter code (h,o,n,s,p,a) is used.

```
Consensus ( ali_{ i_seq | seq_} )
```

- returns the string consensus of alignment *ali\_* as projected to the sequence.

Sequence can be specified by its order number in the alignment or by name.

Example displaying conserved residues:

# 2.21.29. Corr

linear correlation function (Pearson's coefficient  ${\bf r}$  )

```
Corr (R_array1, R_array2)
```

– returns the real value of the linear correlation coefficient. Probability of the null hypothesis of zero correlation is stored in r\_out .

Examples:

See also: LinearFit() function.

# 2.21.30. Cos

cosine function. Arguments are assumed to be in degrees.

```
Cos({ r_Angle | i_Angle })
```

- returns the real value of cosine of its real or integer argument.

Cos ( rarray )

- returns rarray of cosines of each component of the array.

Examples:

# 2.21.31. Cosh

hyperbolic cosine function.

Cosh ( {  $r_Angle | i_Angle \}$  ) – returns the real value of hyperbolic cosine of its real or integer argument.  $Cos(x)=0.5(e^{iz}+e^{-iz})$ 

Cosh (*rarray*) – returns rarray of hyperbolic cosines of each component of the array.

Examples:

show Cosh(1.)	# 1.543081
show Cosh(1)	# the same
show Cosh({-1., 0., 1.})	# returns {1.543081, 1., 1.543081}

# 2.21.32. Count

function creates an iarray.

Count ([i\_Min,] $i_Max$ ) – returns iarray of numbers growing from  $i_Min$  to  $i_Max$ . The default value of  $i_Min$  is 1.

Examples:

```
show Count(-2,1)  # returns {-2,-1,0,1}
show Count(4)  # returns {1,2,3,4}
```

See also the Iarray().

Count (*array*)

- returns iarray of numbers growing from 1 to the number of elements in the array.

# 2.21.33. Deletion

Deletion ( rs\_Fragment, ali\_Alignment [, seq\_fromAli ] [, i\_addFlanks ] [{"all"|"nter"|"cter"|"loop"}] )

- returns the residue selection which flanks deletion points from the viewpoint of other sequences in the *ali\_Alignment*. If argument *seq\_fromAli* is given (it must be the name of a sequence from the alignment), all the other sequences in the alignment will be ignored and only the pairwise subalignment of *rs\_Fragment* and *seq\_fromAli* will be considered. The alignment must be linked to the object. With this function (see also Insertion function) one can easily and quickly visualize and/or extract all *indels* in the three-dimensional structure. The default *i\_addFlanks* parameter is 1. String options:

- "all" (default: no string option) select deletions of all types
- "nter" select only N-terminal fragments
- "cter" select only C-terminal fragments
- "loop" select only the internal zones of deleted loops

See example coming with the Insertion() function description.

#### 2.21.34. Det

determinant function.

Det (matrix)

- returns a real determinant of specified square matrix.

Examples:

```
a=Rot({0. 0. 1.}, 30.) # Z-rotation matrix by 30 degrees
print Det(a) # naturally, it is equal to 1.
```

#### 2.21.35. Disgeo

Solves the so called "DIStance GEOmetry" problem (finding coordinates from a distance set). This function can be used to visualize in two or three dimensions a distribution of homologous sequences:

```
group sequence sel se2 se2 se4 mySeqs
align mySeqs
distMatr=Distances(mySeqs)
```

or any objects between which one can somehow define pairwise distances. Since principal coordinates are sorted according to their contribution to the distances and we can hardly visualize distributions in more than three dimensions, the first two or three coordinates give the best representation of how the points are spread in n-1 dimensions. Another application is restoring atomic coordinates from pairwise distances taken from NMR experiments.

Disgeo(matrix)

- returns matrix [1:n, 1:n] where the each row consists of n-1 coordinates of point [i] sorted according to the eigenvalue (hence, their importance). The first two columns, therefore, contain the two most significant coordinates (say X and Y) for each of n points. The last number in each row is the eigenvalue [i]. If distances are Euclidean, all the eigenvalues are positive or equal to zero. The eigenvalue represents the "principal coordinate" or "dimension" and the actual value is a fraction of data variation due to the this particular dimension. Negative eigenvalues represent "non-Euclidean error" in the initial distances.

Example:

```
read sequences "zincFinger" # read sequences from the file,
list sequences # see them, then ...
group sequence alZnFing # group them, then ...
align alZnFing # align them, then ...
a=Distance(alZnFing) # a matrix of pairwise distances
n=Nof(a) # number of points
b=Disgeo(a) # calculate principal components
corMat=b[1:n,1:n-1] # coordinate matrix [n,n-1] of n points
eigenV=b[1:n,n] # vector with n sorted eigenvalues
xplot= corMat[1:n,2]
plot xplot yplot CIRCLE display # call plot a 2D distribution
```

# 2.21.36. Distance

generic distance function. Calculates distances between two ICM-shell objects or molecular objects, or extracts distances from complex ICM-shell objects.

#### **Distance iarray**

Distance(iarray1, iarray2)

- returns the real sqrt of sum of  $(II_i - I2_i)^2$ .

#### **Distance rarray**

Distance  $(R_X, R_Y)$  – returns the real Cartesian distance between two vectors of the same length. **D** = Sum $((X_i - Y_i)^2)$ 

#### Distance as\_

Distance ( $as_1, as_2$ ) – returns the real distance in Angstroms between centers of mass of the two specified selections. The interactive usage of this function:

- Display your molecule
- type Dist, press TAB
- Ctrl-RightMB click on the atom you want (or double click for a residue) and press RETURN

#### Distance as\_ rarray

Distance (*as\_1*, as\_2, *rarray*) – returns the rarray of distances in Angstroms between the two specified selections containing the same number of atoms (1–1, 2–2, 3–3, ...).

#### **Distance matrix**

Distance ( $M_{coor}$ ) – returns the square matrix of distances between the rows of the input matrix  $M_{coor}$ . Each row contains m coordinates (3 in 3D space). For example: Distance(Xyz(a\_//ca)) returns a square matrix of Ca–Ca distances.

#### **Distance matrix**

Distance ( $M_coorl M_coor2$ ) – returns the matrix of distances between the rows of the two input matrices. Each matrix row may contain any number of coordinates coordinates (3 in 3D space).

For example: **Distance**(**Xyz**(**a**\_/**1:5**/**ca**) **Xyz**(**a**\_/**10:12**/**ca**) returns a 5 by 3 matrix of distances between Ca–s of the two fragments.

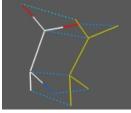
#### **Distance tether**

Distance( $a_{//T}$ )

- returns the real array of distances of the tethered atoms from their target points. Tethers are assumed to be already set, see command set tether. Also note, that the expression Distance( as\_out ) will give the same results if as\_out selection was not changed by another operation; see also special selections.

Example (totally artificial):

```
# 6 commands to create 2 object connected with tethers
build string "se ala"
cp a_ "b"
set tether a_//!h* a_b.
display a_*.
display tether
connect a_ # move it
# output the distances
Distance(a_//T)
#>R
  1.677
  1.493
  1.386
  1.435
  1.645
  1.570
  2.165
  1.399
```



#### **Distance Dayhoff**

Distance (sequence1, *sequence2*) – returns the real measure of similarity between two aligned sequences. Zero distance means 100% identity. The distance is calculated by the following two steps:

1. d1 = 1.0 – (nResidueIdentities/Min(Length(Seq1), Length(Seq2)) (d1 belongs to [0.,1.] range) 2. Distance(Seq1,Seq2) = DayhoffTransformation( d1 )

Transformation practically does not change small distances d1, whereas large distances, especially above 0.9 (10% sequence identity) are increased to take occasional reversals into account. Distances d1 within [0.9,1.0] are transformed to [5.17, 10.] range.

#### **Distance alignment**

Distance (alignment) - returns matrix of pairwise sequence-sequence distances in the alignment.

Example:

```
read alignment msf "azurins"  # read azurins.msf
NormCoord = Disgeo(Distance(azurins))  # 2D sequence diversity in
```

#### Distance two alignments

Distance (*ali\_1 ali\_2* [exact]) – returns the real distance between two alignments formed by the same sequences.

The distance is defined as a number of non-gap columns identical between two alignments.

Two different normalizations are available:

The default normalization is to the shorter alignment. (Distance ( $ali_1 ali_2$ )). In this case the number of equivalent pairs is calculated and is divided by the total number of aligned pairs in the shorter alignment. *This method detects alignment shifts* but does not penalize un-alignment of previously aligned residue pairs. **D** = (**La\_min – N\_commonPairs**)/**La\_min** In the following alignment the residue pairs which are alignments are the same, therefore the distance is 0.

```
show a1 # La1 = 3
ABC---XYZ
ABCDEF---
show a2 # La2 = 6
ABCXYZ
ABCDEF
Distance(a1,a2) # a1 is a subalignment of a2, distance is 0.
0.
```

**exact option: normalization to the longer alignment.** By *longer* we mean the larger number of aligned pairs regardless of alignment length (the latter includes gaps and ends).  $\mathbf{D} = (\mathbf{La}_{max} - \mathbf{N}_{commonPairs})/\mathbf{La}_{max}$  Now in the above example,  $\mathbf{La}_{max} = 6$ , while N\_commonPairs = 3, the distance is 0.5 (e.g. the alignments are 50% different).

Distance(al,a2,exact)  $\mbox{ \# returns 0.5 for the above al and a2 }$ 

Example showing the influence of gap parameters:

```
read sequence msf "azurins.msf"
gapOpen =2.2
a=Align(Azu2_Metj Azup_Alcfa)  # the first alignment
gapOpen =1.9  # smaller gap penalty and ..
b=Align(Azu2_Metj Azup_Alcfa)  # the alignment changes
show 100*Distance(a b )  # 20% difference
show 100*Distance(a b exact )  # 21.7% difference
show a b
```

# 2.21.37. Eigen

eigenvalues/eigenvectors function.

 $Eigen(M_)$ 

- returns the square matrix of eigenvector columns of the input symmetric square matrix  $M_{-}$ . Eigenvalues sorted by their values are stored in the R\_out rarray.

Example:

```
A[2,2] = 4.
for i = 1, 3-1
                       # the matrix must be symmetric
 for j = i+1, 3
   A[i,j] = A[j,i]
 endfor
endfor
X = Eigen(A)
                       # calculate eigenvectors...
V = R \text{ out}
                       # and save eigenvalues in rarray V
printf "eigenvalue 1 eigenvalue 2 eigenvalue 3\n"
printf "%12.3f %12.3f %12.3f\n", V[1], V[2], V[3]
printf "eigenvector1 eigenvector2 eigenvector3\n"
for i = 1, 3
 printf "%12.3f %12.3f %12.3f\n", X[i,1], X[i,2], X[i,3]
endfor
```

# 2.21.38. Energy

function.

Energy (string)

- returns the real sum of **pre-calculated** energy and penalty (i.e. geometrical restraints) terms specified by the string.

**Important**: the terms must be **pre-calculated** by invoking one of the following commands where energy is calculated at least once: show energy, minimize, ssearch command and montecarlo command.

#### Note:

- Allowed terms in the string are "vw, 14, hb, el, to, af, bb, bs, cn, tz, rs, xr, sf";
- "func" stands for the total of all the terms, both energy and penalty;
- "ener" is only the energy part (i.e. "vw, 14, hb, 31, to, af, bb, bs, sf");
- "pnlt" is only the penalty part (i.e. "cn,tz,rs,xr").
- load conf and load frame commands fill out all the energy/penalty terms, which are stored in both stacks and movies (of course the values also depend on a set of free variables). You can get the energy/penalty terms of the loaded conformation without explicitly recalculating them.

```
build
show energy
print Energy("vw,14,hb,el,to") # ECEPP energy
read stack "f1"
load conf 0
print Energy("func") # extract the best energy without recalculating it
```

```
Energy ( conf i_confNumber)
```

- returns the table of all the energy components for the a given stack conformations.

The table has two arrays:

- sarray of the energy term names (.hd) and
- rarray of energy values for each energy term ( .ey ) and

```
Energy({ stack | conf })
```

- returns the rarray of total energies of stack conformations. Useful for comparison of spectra from different simulations.

Examples:

```
set terms only "vw,14,hb,el,to" # set energy terms
show energy v_//xi*
                                # calculate energy with only
                                # side chain torsions unfixed
          # energy depends on what variables are fixed since
          # interactions inside rigid bodies are not calculated,
          # and rigid body structure depends on variables
                        # a is equal to the sum of two terms
a = Energy("vw, 14")
electroMethod="MIMEL"
                       # MIMEL electrostatics
set terms only "el,sf" # set energy terms
show energy
print Energy("ener")
                       # total energy
print Energy("sf")
                       # only the surface part of the solvation energy
                       # electrostatic energy
print Energy("el")
print r_out
                       # electrostatic part of the solvation energy
```

#### 2.21.39. Error

function indicates that the previous ICM-shell command has completed with error.

Error

- returns logical yes if there was an error in a previous command (not necessarily in the last one). After this call the internal error flag is reinstalled to no.

Error(string)

- returns string with the last error message. It also returns integer code of the *last* error in your script in i\_out. In contrast to the logical Error() function, here the internal error code is **not** reinstalled to 0, so that you can use it in expressions like if( Error ) print Error(string).

Examples:

read pdb "1mng" # this file contains strange 28-th residue if (Error) print "These alternative positions will kill me" read pdb "labcd" # file does not exist errorMessage = Error(string) ier = i\_out if ier == 3102 print "What kind of name is that?" See also: errorAction, s skipMessages, l warn, Warning

Error(r\_x[reverse])

- returns real complementary error function of real x : erfc(x)=1.-erf(x)), defined as

(2/sqrt(pi)) integral{x to infinity} of  $exp(-t^2)$  dt

or its inverse function if the option reverse is specified. It gives the probability of a normally distributed (with mean 0. and standard deviation 1./Sqrt(2.)) value to be larger than  $r_x$  or smaller than  $-r_x$ .

Examples:

show 1.-Error(Sqrt(0.5)) # P of being inside +-sigma (about 68%) show Error(2.\*Sqrt(0.5)) # P of being outside +- 2 sigma

Error ( $R_x$ )

- returns rarray of *erfc(x)=1.-erf(x)*) functions for each element of the real array (see above).

Examples:

```
x=Rarray(1000 0. 5. )
plot display x Error(x ) {0. 5. 1. 1. 0. 1. 0.1 0.2 }
plot display x Log(Error(x ),10.) {0. 5. 1. 1.}
    #NB: can be approximated by a parabola
    #to deduce the appr. inverse function.
    #Used for the Seq.ID probabilities.
```

#### 2.21.40. Exist

function indicates if an ICM-entity exists or not.

Exist (s\_fileName) - returns logical yes if the specified file or directory exists, no otherwise.

Exist (key, *s\_keyName*) – returns logical yes if the specified keystoke has been previously defined. See also: set key command.

Exist (object) - returns logical yes if there is at least one molecular object in the shell, no otherwise.

```
Exist (view) - returns logical yes if the GL - graphics window is activated, no otherwise.
```

Exist (gui) - returns logical yes if the GRAPHICS USER INTERFACE menus is activated, no otherwise.

Examples:

```
if (!Exist("/data/pdb/") then
    unix mkdir /data/pdb
endif
if(!Exist(key,"Ctrl-B")) set key "Ctrl-B" "l_easyRotate=!l_easyRotate"
if !Exist(gui) gui simple
```

#### 2.21.41. Existenv

function indicating if an UNIX-shell environmental variable exists.

Existenv (s\_environmentName)

- returns logical yes if the specified named environment variable exists.

Example:

if(Existenv("ICMPDB")) s\_pdb=Getenv("ICMPDB")

See also: Getenv(), Putenv().

#### 2.21.42. Extension

function.

Extension(string[dot])

- returns string which would be the extension if the string is a file name. Option dot indicates that the dot is excluded from the extension.

```
Extension(sarray[dot])
```

- returns sarray of extensions. Option dot indicates that the dot is excluded from the extensions.

```
print Extension("aaa.bbb.dd.eee")  # returns ".eee"
show Extension({"aa.bb","122.22"} dot)  # returns {"bb","22"}
read sarray "filelist"
if (Extension(filelist[4])==".pdb") read pdb filelist[4]
```

# 2.21.43. Exp

exponential mathematical function  $(e^x)$ .

Exp(real)

- returns the real exponent.

Exp(rarray)

- returns rarray of exponents of rarray components.

Exp ( matrix )

- returns matrix of exponents of *matrix* elements.

Examples:

```
print Exp(deltaE/(Boltzmann*temperature))  # probability
print Exp({1. 2.})  # returns { E, E squared }
```

#### 2.21.44. Field

function.

```
Field(s_[s_precedingString]i_fieldNumber[s_fieldDelimiter])
```

- returns the specified field. Parameter s\_fieldDelimiter defines the separating characters (space and tabs by default). If the field number is less than zero or more than the actual number of fields in this string, the function returns an empty string.

#### The s\_fieldDelimiter string

Single character delimiter can be specified directly, e.g.

Field("a b c",3," ") # space
Field("a:b:c",3,":") # colon

Alternative characters can be specified sequentially, e.g.

```
Field("a%b:c",3,"%:") # percent OR colon
```

*Multiple* occurence of a delimiting character can be specified by *repeating* the same character *two* times, e.g.

```
Field("a b c",3," ") # two==multiple spaces in field delim
Field("a%b::::c",3,"%::") # a single percent or multiple colons
```

You can combine a single-character delimiters and multiple delimiters in one *s\_fieldDelimiter* string.

More examples:

```
Field(S_,[s_precedingString]i_fieldNumber[s_fieldDelimiter])
```

- returns an string array of fields selected from *S*\_string array.s\_fieldDelimiter is the delimiter. If the field number is less than zero or more than the actual number of fields in this string, an element of the array will be an empty string.

Examples:

See also: Split().

# 2.21.45. User field from a selection

Field(as\_)

```
Field({ rs_|ms_|os_} [ i_fieldNumber ])
```

returns rarray of user-defined field values of a selection.

Atoms. Only one user defined field can be set to atoms, e.g.

```
set field a_//* Random(0.,1.,Nof(a_//*)) show Field( a_//* )
```

#### Residues, molecules and objects.

Three user fields can be defined for each residue and up to 16 for molecules and objects. To extract them specify  $i_{fieldNumber}$ . The level of the selection determines if the values are extracted from residues, molecules or objects. Use the selection level functions Res Mol and Obj to reset the level if needed. For example: Res(Sphere(gg, a\_1. 3.)) selects residues of the 1st object which are closer than 3. A to grob gg. Example:

```
set field 2 a_/A Random(0.,1.,Nof(a_/A)) \# set the 2nd field to random values color a_/* Field( a_/A 2 ) \# color by it
```

See also: set field, Smooth to 2D or 3D averate user fields, Select to select by user defined field.

### 2.21.46. File

function returning file names or attributes of named files.

File (*os*) returns the name of the source file for this object. If the object was created in ICM or did not come from an object or PDB file, it returns an empty string.

#### Example:

```
read pdb "/home/nerd/secret/hiv.ob"
File( a_ )
    /home/nerd/secret/hiv.ob
```

File(s\_file\_or\_dir\_Name)

- returns string with the file or directory attributes separated by space. If file or directory do not exist the function returns "- - - 0" Otherwise, it contains the following 4 characters separated by space and the file size:

- 1. type character:
  - ♦ 'f' regular file
  - ♦ 'd' directory
  - ♦ 'l' symbolic link
  - ◆ 'c' character special file
  - ♦ 'p' pipe
- 2. 'r' if you can read the file (or from the directory)
- 3. 'w' if you can write to this file (or directory)
- 4. 'x' if you can execute this file (or cd to this directory)
- 5. file size in bytes

To get a string with any field use Field(File(*s\_name*), *i\_fieldNumber*). To get the size, use Integer(Field(File(*s\_name*),4)).

Example:

```
if ( Integer(Field(File(s_name),4)) < 10 ) return error "File is too small"
```

#### 2.21.47. Find

function searching all fields (arrays) of a table.

Find ( table s\_searchWords )

- returns table containing the entries matching all the words given in the *s\_searchWords* string.

If *s\_searchWords* is "word1 word2" and table contains arrays a and b this "all text search" is equivalent to the expression :

```
(t.a=="word1" | t.b == "word1") (t.a=="word2" | t.b == "word2").
```

Examples:

```
read database "ref.db" # database of references
group table ref $s_out # group created arrays into a table
show Find(ref,"energy profile") ref.authors == "frishman"
```

#### 2.21.48. Floor

rounding function.

Floor ( r\_real [ r\_base ] )

- returns the largest real multiple of *r\_base* not exceeding *r\_real*.

Floor (R\_real [r\_base])

– returns the rarray of the largest multiples of  $r_{base}$  not exceeding components of the input array  $R_{real}$ .

Default  $r\_base = 1.0$ .

See also: Ceil().

#### 2.21.49. Getenv

function returning value for an environment name.

Getenv ( s\_environmentName )

- returns a string of the value of the named environment variable.

Example:

```
user = Getenv("USER")  # extract user's name from the environment
if (user=="vogt") print "Hi, Gerhard"
```

See also: Existenv(), Putenv().

# 2.21.50. Gradient

function.

Gradient()

- returns the real value of the root-mean-square gradient over free internal variables.

Gradient(vs\_var)

- returns the rarray of pre-calculated energy derivatives with respect to specified variables.

Gradient (*as*\_|*rs*\_)

- returns the rarray of pre-calculated energy derivatives with respect to atom positions  $(G[i] = Sqrt(Gxi^*Gxi+Gyi^*Gyi+Gzi^*Gzi))$ 

The function returns atom–gradients for atom selection ( $as_{-}$ ) or average gradient per selected residue, if residue selection is specified ( $rs_{-}$ ).

You can display the actual vectors/"forces" (-Gxi, -Gyi, -Gzi) by the display gradient command.

**Important**: to use the function, the gradient must be pre-calculated by one of the following commands: show energy, show gradient, minimize.

Examples:

```
show energy \# to calculate the gradient and its components
if (Gradient() > 10.) minimize
show Max(Gradient(a_//c*) \# show maximum "force" applied to the carbon atoms
```

#### 2.21.51. Grob

function to generate graphics objects.

Grob ("arrow", {  $R_3 | R_6$  })

- returns grob containing 3D wire arrow between either  $\{0., 0., 0.\}$  and  $R_3$ , or between  $R_6[1:3]$  and  $R_6[4:6]$ .

Grob("ARROW", { *R*\_3 | *R*\_6 })

- returns grob containing 3D solid arrow. You may specify the number of faces by adding integer to the string: e.g. "ARROW15" (rugged arrow) or "ARROW200" (smooth arrow).

See also: GROB.relArrowSize.

- returns grob containing a wire parallelepiped for a given cell.

If only  $R_3$  is given, angles {90., 90., 90.} are implied.

Grob("CELL", { *R\_3* | *R\_6* })

- returns grob containing a solid parallelepiped for a given cell.

If only  $R_3$  is given, angles {90., 90., 90.} are implied.

Example:

```
read csd "qfuran"
gcell = Grob("CELL",Cell())  # solid cell
display a_//* gcell transparent # fancy stuff
```

Grob ( "label", R\_3, s\_string )

- returns grob containing a point at  $R_3$  and a string label.

Grob("line",  $R_3N$ )

- returns grob containing a polyline R\_3N[1:3], R\_3N[4:6], ...

Example:

```
display a__crn.//ca,c,n
g = Grob("line",{0.,0.,5.,5.,5.})  # a simple line (just as an example)
display g yellow
gCa = Grob("line",Rarray(Xyz(a_//ca)))  # connect Cas with lines
display gCa pink  # display the grobs
```

Grob ( "SPHERE", r\_radius i\_tesselationNum )

- returns grob containing a solid sphere. The *i\_tesselationNum* parameter may be 1,2,3. (do not go too high).

Example:

```
Grob ( grob R_6rgbLimits )
```

returns a grob containing selection of vertices of the source grob. The vertices with colors between the RGB values provided in the 6-dim. array of limits will be selected. The array of limits consists of real

numbers between 0. and 1. : { *from\_R, to\_R, from\_G, to\_G, from\_B, to\_B* }

If you want a limit to be outside possible rgb values, use negative numbers of numbers larger than 1., e.g. a selection for the red color could be:  $\{0.9, 1., -0.1, 0.1, -0.1\}$ 

The grob created by this operation has a limited use and will contain only vertices (no edges or triangles). This form of the Grob function can be used to find out which atoms or residues are located to spots of certain color using the Sphere( grob as\_) function.

Example:

```
buildpep "ADERD" # a peptide
dsRebel a_ no no
g=Grob(g_skin {0.9,1.,-0.1,0.1,-0.1,0.1}) # red color
display g_skin transparent
display g
show Res(Sphere( g, a_//* 1.5))
```

# 2.21.52. Histogram

function to create a histogram of an array. Function returns matrix [n,2], where *n* is number of cells, the first row contains a number of elements in each cell and the second row contains mid-points of each cell.

Histogram (I\_inputArray )

- returns matrix with a histogram of the input array.

Histogram(R\_inputArray, i\_numberOfCells)

- returns histogram matrix [*i\_numberOfCells,2*] in which the whole range of the *R\_inputArray* array is equally divided in *i\_numberOfCells* windows.

Histogram(R\_inputArray, r\_cellSize)

- returns matrix [n,2], dividing the whole range of *R\_inputArray* equally into *r\_cellSize* windows.

Histogram(R\_inputArray, r\_from, r\_to, r\_cellSize)

- returns matrix [ n,2], dividing into equal cells of r\_cellSize between minimum value, maximum value.

Histogram(R\_inputArray, R\_cellRuler)

- returns matrix [n,2], dividing the range of the input array according to the  $R_cellRuler$  array, which must be monotonous.

```
plot display Histogram({ -2, -2, 3, 10, 3, 4, -2, 7, 5, 7, 5}) BAR
a=Random(0. 100. 10000)
u=Histogram(a 50)
```

```
s_legend={"Histogram at linear sampling curve" "Random value" "N"}
plot display regression BAR u s_legend
a=Random(0. 100. 10000)
b=.04*(Count(1 50)*Count(1 50))
u=Histogram(a b)
s_legend={"Histogram at square sampling curve" "Random value" "N"}
plot display BAR u s_legend
b=Sqrt(100.*Count(1 100))
s_legend={"Histogram at square root sampling curve" "Random value" "N"}
plot display green BAR Histogram(a b) s_legend
```

#### 2.21.53. larray

function to create/declare an empty iarray or transform to an iarray.

```
Iarray(i_NumberOfElements[i_value])
```

- returns iarray of *i\_NumberOfElements* elements set to *i\_value* or zero. You can also create an zero-size integer array: Iarray(0).

Iarray(rarray)

- returns iarray of integers nearest to real array elements in the direction of the prevailing rounding mode magnitude of the real argument.

Iarray (sarray) - converts sarray into an iarray.

Examples:

```
a=Iarray(5)# returns {0 0 0 0 0}a=Iarray(5,3)# returns {3 3 3 3}b=Iarray({2.1, -4.3, 3.6})# returns {2, -4, 4}c=Iarray({"2", "-4.3", "3.6"})# returns {2, -5, 3}
```

Iarray(iarray reverse)

- converts input real array into an iarray with the reversed order of elements. Example:

 $Iarray(\{1 \ 2 \ 3\} reverse) \# returns \{3 \ 2 \ 1\}$ 

See also: Sarray(S\_reverse), Rarray(S\_reverse), String(0,1,s)

#### 2.21.54. larray( stack ): numbers of visits for all stack conformations

larray(stack) - returns the iarray of the numbers of visits for each stack conformation. This is the same number as shown by the nvis> line of the show stack command. Example:

show sta	ack				
iconf>	1	2	3	4	5
ener>	-15.3	-15.1	-14.9	-14.8	-13.3
rmsd>	84.5	75.3	6.4	37.2	120.8
naft>	3	0	4	0	2

nvis> 10 9 8 1 4 Integer(stack) # returns { 10 9 8 1 4 }

# 2.21.55. IcmSequence

creates a "sequence" for an ICM molecular object. Output is in icm.se -file format.

IcmSequence ( { sequence | string | rs\_ }, [ s\_N-Term, s\_C-Term ] )

- returns multiline string with full (3-char.) residue names which may be a content of an icm.se file. The source of the sequence may be one of the following:

- a sequence, e.g. IcmSequence(lcrn\_m)
- a string, e.g. "ASDGFRE", or "SfGDA; WER".
- or residue selection, rs\_, (e.g. a\_2,3/\*).

Rules for one-letter coding:

- standard L amino-acids: upper case one-letter code (B,J,X,Z are illegal), e.g. ACD
- D-amino acids: lower case for a corresponding amino acid (e.g. AaA for ala Dala ala)
- new molecule: use **semicolon** or **dot** as a chain separator (;) (e.g. AAA; WWW)

The N-terminal and C-terminal groups will be added if their names are explicitly specified. To skip a terminal group on one end only, provide the empty string (e.g. "","cooh").

The resulting strin can be saved to a ICM mol-sequence file and further edited for unusual amino-acids (see icm.res).

Examples:

There is a convenient macro called buildpep to create a single or multiple chain peptides. Example:

buildpep "SDSRAARESW;KPLKPHYATV" # two 10-res. peptides

See also icm.se for a detailed description of the ICM-sequence file format.

#### 2.21.56. Index

function.

2.21.55. IcmSequence

Index({ s\_source | seq\_source }, { s\_pattern | seq\_pattern }, [ { last | i\_skip ] )

- returns integer value indicating the position of the pattern substring in the source string, or 0 otherwise. Option last returns the index of the last occurence of the substring.

The  $i\_skip$  argument starts search from the specified position in the source string, e.g. Index("words words","word",3) returns 7. If  $i\_skip$  is negative, it specifies the number of characters from the end of the string in which the search is performed.

Examples:

```
show Index("asdf","sd") # returns 2
show Index("asdf" "wer") # return 0
a=Sequence("AGCTTAGACCGCGGAATAAGCCTA")
show Index(a "AATAAA") # polyadenylation signal
show Index(a "CT" last) # returns 22
show Index(a "CT" 10) # starts from position 10. returns 22
show Index(a "CT", -10) # search only the last 10 positions
```

Another example in which we output all positions of all –"xxx.." stretches in a sequence " xxxx xxxx xxxx xxxx ... xxxx " (must end with space)

```
Index ( sarray, string )
```

- returns integer value indicating the *sarray* element number exactly matching the *string*, or 0 otherwise.

Examples:

```
show Index({"Red Dog","Amstal","Jever"}, "Jever")  # returns 3
show Index({"Red Dog","Amstal","Jever"}, "Bitburger")  # returns 0
```

Index(object)

- returns integer value of sequential number of the current object in the molecular object list, or 0 if no objects loaded. (Note that here object is used as a keyword.)

```
l_commands = no
read pdb "lcrn"
read object "crn"
printf "The object a_crn. is the %d-nd, while ...\n", Index(object)
set object a_1.
```

printf "the object a\_lcrn. is the  $d-st.\n"$ , Index(object)

#### 2.21.57. Indexx

function to find location of substring pattern.

Indexx ( { string | sequence }, s\_Pattern )

- returns an integer value indicating the position of the *s\_Pattern* (see pattern matching) in the string, or 0 otherwise. Allowed meta-characters are the following:

- \* any string including an empty string;
- ? any single character;
- [ *string* ] any of the enclosed characters;
- [! string ] any but the enclosed characters.
- ^ beginning of a string
- \$ string end

Examples:

```
      show Indexx("asdf","s[ed.]")
      # returns 2

      show Indexx("asdfff","ff$")
      # returns 5 (not 4)

      show Indexx("asdf" "w?r")
      # return 0
```

#### 2.21.58. Insertion

function selecting inserted residues.

Insertion (rs\_Fragment, ali\_Alignment [, seq\_fromAli ][, i\_addFlanks ] [{"all"|"nter"|"cter"|"loop"}])

- returns the residue selection which form an insertion from the viewpoint of other sequences in the *ali\_Alignment*. If argument *seq\_fromAli* is given (it must be the name of a sequence from the alignment), all the other sequences in the alignment will be ignored and only the pairwise subalignment of *rs\_Fragment* and *seq\_fromAli* will be considered. The alignment must be linked to the object. With this function (see also Deletion() function) one can easily and quickly visualize all indels in the three-dimensional structure. The default *i\_addFlanks* parameter is 0.

String options:

- "all" (or no string option) select insertions of all types
- "nter" select only N-terminal fragments
- "cter" select only C-terminal fragments
- "loop" select only the internal loops

#### Examples:

read pdb "1phc.m/" # read the first molecule form this pdb-file

#### 2.21.57. Indexx

```
read pdb "2hpd.a/" \  \  \, \# do the same for the second molecule make sequences a_*. \  \  \, \# you may also read the sequence and
                        # the alignment from a file
                        # on-line seq. alignment.
aaa=Align( )
                         # You may read the edited alignment
                         # worm representation
assign sstructure a *. " "
display ribbon
link a_*. aaa
                        # establish connection between sequences and 3D obj.
superimpose a_1. a_2. aaa
display ribbon a_*.
color a_1. ribbon green
color ribbon Insertion(a_1.1 aaa) magenta
color ribbon Insertion(a_2.1 aaa) red
show aaa
```

#### 2.21.59. Integer

function converting to integer type.

Integer(r\_toBeRounded)

- returns the integer nearest to real *r\_toBeRounded* in the direction of the prevailing rounding mode magnitude of the real argument.

Integer ( rarray ) - see Iarray ( rarray ).

Integer (*string*) – converts string into integer, ignores irrelevant tail. Reports error if conversion is impossible.

Examples:

```
show Integer(2.2), Integer(-3.1)  # 2 and -3
jj=Integer("256aaa")  # jj will be equal to 256
```

```
See also: Iarray().
```

Integer (*iarray*) transforms integer array containing only one element into an integer. You can also convert a one element array into an integer with the Sum() or the Mean() functions. If there are more then one elements, the first element is taken.

#### 2.21.60. Integral

function.

Integral (*rarray*  $r_x$ *Increment*) – calculates the integral rarray of the function represented by *rarray* on the periodically incremented abscissa x with the step of  $r_x$ *Increment*.

Integral (rarrayY rarrayX) – calculates the integral rarray of the function represented by rarrayY on the set of abscissa values rarrayX.

Let us integrate  $3 x^2 - 1$ , determined on the rarray of unevenly spaced *x*. The expected integral function is  $x^3 - x$ 

```
x=Rarray(100 ,-.9999, .9999 )
x=x*x*x
plot display x Integral((3*x*x-1.) x) cross
```

#### 2.21.61. Interrupt

function.

Interrupt

- returns logical yes if ICM-interrupt (Ctrl-Backslash, ^\) has been received by the program. Useful in scripts and macros.

Examples:

if (Error | Interrupt) return

#### 2.21.62. Label

function returning a molecular label string.

Label(rs\_)

- returns sarray of residue labels of the selected residues *rs*\_composed according to the resLabelStyle preference.

See also: Name function (returns residue names).

Label ( os\_objects )

- returns sarray of long names of selected objects.

See also: Name function which returns the regular object names and the most detailed chemical names of compounds.

Label(vs\_var)

2.21.61. Interrupt

- returns sarray of labels of selected variables.

Examples:

```
build
resLabelStyle = "Ala 5"  # other styles also available
aa = Label(a_/2:5)  # extract residue name and/or residue number info
show aa  # show the created string array
```

# 2.21.63. Length

function.

Length ( { string | matrix | sequence | alignment | profile } )

- returns integer length of specified objects.

Length ( sarray )

- returns iarray of length of strings elements of the sarray.

Length ( { iarray | rarray } )

- returns the real vector length (distance from the origin for a specified vector Sqrt(Sum(I[i]\*I[i])) or Sqrt(Sum(R[i]\*R[i])), respectively).

Examples:

<pre>len=Length("asdfg")</pre>	# len is equal to 5
a=Matrix(2,4)	# two rows, four columns
nCol=Length(a)	# nCol is 4
read profile "prof"	# read sequence profile
show Length(prof)	# number of residue positions in the profile
<pre>vlen=Length({1 1 1})</pre>	# returns 1.732051

# 2.21.64. LinearFit

the linear regression function.

LinearFit(R\_X, R\_Y, [R\_Errors])

- returns the A,B,StdDev,Corr 4-dimensional array of the parameters of the linear regression for a scatter plot Y(X):  $R_Y = A * R_X + B$ , where the slope A and the intercept B are the first and the second elements, respectively. The third element is the standard deviation of the regression, and the fourth is the correlation coefficient. Residuals  $R_Y - (A * R_X + B)$  are stored in the R\_out array.

You can also provide an array of expected errors of  $R_Y$ . In this case the weighted sum of squared differences will be optimized. The weights will be calculated as:

 $W_i = 1/R\_Errors_i^2$ 

Example:

```
X = Random(1., 10., 10)
Y = 2.*X + 3. + Random(-0.1, 0.1, 10)
lfit = LinearFit(X Y)
printf "Y = %.2f*X + %.2f\n", lfit[1],lfit[2]
printf "s.d. = %.2f; r = %.3f\n", lfit[3],lfit[4]
show column X, Y, X*lfit[1]+lfit[2], R_out
```

A more complex linear fit between a target set of  $Y_i$ , i=1:n values and several parameters  $X_{i,j}$ (i=1:n,j=1:m) potentially correlating with  $Y_i$  is achieved in 3 steps:

M1=Transpose(X)\*X
M2=Power(M1,-1)
W =(M2\*Transpose(X))\*Y

The result of this operation is vector of weights W for each of m components.

Now you can subtract the predicted variation from the initial vector ( $Y_2 = Y - X^*W$ ) and redo the calculation to find  $W_2$ , etc. A proper way of doing it, however, is to calculate the eigenvalues of the covariance matrix.

# 2.21.65. Link

the linked SWISSPROT sequences function.

Link (*ms*) – returns the sarray of the swissprot sequence IDs (e.g. {"PHEA\_HUMAN"}) referenced by the selected molecules. These references (or links) are read from the pdb entries.

# 2.21.66. Log

the logarithm function.

Log (real) - returns the real natural logarithm of a specified positive argument.

 $\log (real r_realBase)$  – returns the real logarithm of a specified positive argument (e.g. the base 10 logarithm is Log(x, 10)).

Log(rarray) – returns an rarray of natural logarithms of the array components (they must not be negative, zeroes are treated as the least positive real number, ca.  $10^{-38}$ ).

Log (*rarray r\_realBase*) – returns an rarray of logarithms of the array components (they must not be negative), arbitrary base.

Log (*matrix* [*r\_realBase*]) – returns a matrix of logarithms of the matrix components (they must not be negative).

# 2.21.67. Map

function.

Map (*m\_map*, *I\_6box* [simple]) – returns map which is a transformation (expansion or reduction) of the input *m\_map* to new *I\_6box* box ({ *iMinX,jMinY,kMinZ,iMaxX,jMaxY,kMaxZ*}).

Examples:

```
read object "crn"
read map "crn"
display a_//ca,c,n m_crn
ml = Map(m_crn, {0 0 0 22 38 38}) # half of the m_crn
m2 = Map(m_crn, {0 0 0 88 38 38}) # double of the m_crn
display m1
display m2
```

#### 2.21.68. Mass

function.

```
Mass ( as_| rs_| ms_| os_)
```

- returns rarray of masses of selected atoms, residues, molecules or objects, depending on the selection level.

Examples:

```
buildpep "ala his trp glu"
objmasses = Mass( a_*.)
molmasses = Mass( a_* )
resmasses = Mass( a_/* )
masses=Mass( a_//!?vt* )  # array of masses of nonvirtual atoms
molweight = Mass( a_1 )  # mol.weight of the 1st molecule
molweight = Sum(Mass( a_1//* ))  # another way to calculate 1st mol.weight
```

See also: Nof( sel atom ), Charge( sel )

# 2.21.69. Matrix

function.

Matrix (i\_NofRows, *i\_NofColumns* [*r\_value*]) – returns matrix of specified dimensions. All components are set to zero or *r\_value* if specified.

Matrix (*i\_size*) – returns square unity matrix of specified size.

Matrix (rarray) - converts vector[1:n] to one-row matrix[1:1,1:n].

#### **Tensor product**

Matrix ( $R_A R_B$ )

- returns tensor product of two vectors or arbitrary dimensions:  $M_{ij} = R_A[i] * R_B[j]$ 

Examples:

Matrix (*rs\_1 rs\_2*) - returns matrix of contact areas. See also: Cad, Area.

Matrix  $(ali_)$  – returns a matrix of normalized pairwise Dayhoff evolutionary distances between the sequences in alignment  $ali_{-}$  (for similar sequences it is equal to the fraction mismatches).

Matrix (ali\_, number) – returns a matrix of alignment. It contains reference residue numbers for each sequence in the alignment, or -1 for the gaps. The first residue has the reference number of 0 (make sure to add 1 to access it from the shell).

#### 2.21.70. Max

maximum-value function.

Max({ rarray | map })

- returns the real maximum-value element of a specified object

Max (*iarray*) – returns the integer maximum-value element of the iarray.

Max(matrix) – returns the rarray of maximum-value element of each column of the matrix. To find the maximum value use the function twice (Max(Max(m)))

Max (integer1, integer2, ...) – returns the largest integer argument.

Max (real1, real2, ...) - returns the largest real argument.

show Max({2. 4. 7. 4.}) # 7. will be shown

# 2.21.71. MaxHKL

an array of three maximal crystallographic h,k,l indices at a given resolution.

MaxHKL ( {  $map_{|os_{|}|} [ R_6CellParameters ]$  },  $r_minResolution ) => I3$ 

the function extracts the cell parameters from  $map_{, os_{object, or reall array of {a,b,c,alpha,beta,gamma}, and calculates an iarray of three maximal crystallographic indices { <math>hMax$ , kMax, lMax } corresponding to the specified  $r_{min}Resolution$ .

#### 2.21.72. Mean

average-value function.

```
Mean({ rarray | map })
```

- returns the real average-value of elements of the specified ICM-shell objects.

Mean(*iarray*)

- returns the real average-value of the elements of the *iarray*.

Mean (*matrix*)

- returns rarray [1:m] of average values for each *i*-th column matrix[1:n,i].

Mean (R1 R2)

- for two real arrays of the same size returns rarray [1:m] of average values for each pair of corresponding elements.

Examples:

#### 2.21.73. Min

minimum-value function.

Min ({ rarray | map }) - returns the real minimum value element of a specified object

Min (*iarray*) – returns the integer minimum-value element of the iarray.

Min (*matrix*) – returns the rarray of minimum–value element of each column of the matrix. To find the minimum value use the function twice (e.g. Min(Min(m)))

Min (integer1, *integer2* ... ) – returns the smallest integer argument.

Min (real1, real2, ...) - returns the smallest real argument.

#### Examples:

show Min({2. 4. 7. 4.}) # 2. will be shown show Min(2., 4., 7., 4.) # 2. will be shown

#### 2.21.74. Money

function to print money figures.

Money ({ *i\_amount* | *r\_amount*}, [ *s\_format*]) – returns a string with the traditionally decorated money figure. *s\_format* contains the figure format and the accompanying symbols.

- %m specification for the rounded integral amount;
- % . m specification to add cents after dot. The default is "\$%.m", i.e. Money(1222.33) returns \$1,222.33.
- %M the same as %m but with dot instead of comma in the European style
- %. M the same as %.m dot and comma are inverted in the European style

Examples:

# 2.21.75. Mod

remainder (module) function. Similar to, but different from Remainder() function:

function	description	example
Mod(x,y)	brings x to [0, y] range	Mod(17.,10.) => 7.
Remainder(x,y)	brings x to $[-y/2, y/2]$ range	Remainder(17.,10.)=>-3.

Mod (i\_divisor, [i\_divider]) - returns integer remainder.

Mod (r\_divisor, [ $r_divider$ ]) – returns the real remainder  $r = x - n^*y$  where n is the integer nearest the exact value of x/y; r belongs to [0, |y|] range.

Mod (iarray, [*i\_divider*]) – returns the iarray of remainders (see the previous definition).

Mod (rarray, [r\_divider]) - returns the rarray of remainders (see the previous definition).

The default divider is 360. (or 360) since we mostly deal with angles.

Examples:

phi = Mod(phi) # transform angle to [0., 360.] range a = Mod(17,10) # returns 7

#### 2.21.76. Mol

molecule function.

Mol ( {  $os_{|} rs_{|} as_{|}$  } ) – selects molecules related to the specified objects  $os_{|}$ , residues  $rs_{|}$  or atoms  $as_{|}$ , respectively.

Examples:

```
show Mol( Sphere(a_1//* 4.) )
    # molecules within a 4 A vicinity of the first one
    # Sphere function Sphere(as_atoms) selects atoms.
```

See also: Atom, Res, Obj.

#### 2.21.77. Name

generic function returing strings or string arrays with names of things.

Name (s\_Path\_and\_Name) - returns file name sub-string if full path is specified

Name ( { command | function | macro | integer | real | string | logical | iarray | rarray | sarray | matrix | map | grob | alignment | table | profile | sequence } ) - returns a string array of object names for the specified class.

Name ({ iarray | rarray | sarray | matrix | map | grob | alignment | table | profile | sequence } select) - returns a string array of names of **selected** objects for the specified class.

Name (*as*\_) – returns sarray of names of selected atoms.

Name (*as*\_sequence) – returns sarray of chemical names of the selected atoms according to the icm.cod file (one–letter chemical atom names are low case, e.g. "c", two–letter names start from an upper–case letter, like "Ca"). The names from the periodic table are used in the wrGaussian macro.

Name ( $rs_{-}$ ) - returns sarray of names of selected residues. To obtain a one-letter code sequence, use Sequence( $rs_{-}$ ) and to convert it to a string use String(Sequence( $rs_{-}$ )).

Name (*ms*\_) – returns sarray of names of selected molecules.

Name (ms\_sequence) - returns sarray of names of sequence linked to the specified molecules ms\_.

Name (os\_) - returns sarray of names of selected objects.

Name (vs\_) – returns sarray of names of selected variables.

Name (*alignment*) – returns sarray of constituent sequence names.

Name (table) - returns sarray or constituent table ICM-shell object names.

Name (sequence) - returns string name of specified sequence.

Examples:

read alignment msf "azurins" # load alignment seqnames = Name(azurins) # extract sequence names show Name( Acc( a\_/\* ) ) # array of names of exposed residues

#### 2.21.78. Namex

long name (or description) function.

Namex(os\_)

- returns sarray of comments of selected objects *os*\_(i.e. a string for each object). This field is set to the chemical compound name by the read pdb command. Alternatively, you can set your own comment with the set comment os\_ s\_comment command. If you have a single object and want to convert a string array of one element (corresponding to this one object) to a simple string, use this expression, e.g.: Sum(Namex(a\_)). Other manipulations with a multiline string can be performed with the Field, Integer, Real, Split functions (see also s\_fieldDelimiter).

Example. We stored values in the comment field in annotations like this: "LogP 4.3\n". Now we extract the values following the "LogP" field name:

read object s\_icmhome + "all" # multiple object file with LogP data
# extract numbers following the 'LogP' word in the object comments
logPs = Rarray(Namex(a\_\*.), "LogP", 1," \n")

Namex (seq\_) - returns string of long name ('description' field in Swissprot).

Example:

```
read index s_inxDir+"/SWISS.inx"
read sequence SWISS[2]  # read the 2nd sequence from Swissprot
show Namex( sequence[0] )
```

#### 2.21.79. Next

selection function.

Next ( { *as*\_| *rs*\_| *ms*\_| *os*\_ } )

- selects atom, residue, molecule, or object immediately following the selected one. Next( the\_last\_element ) returns an empty selection.

Examples:

show Next(  $a_/4$  ) ) # show residues number 5 phi5=Torsion(  $a_/5/c$  ) # psi5 formally belongs to the next residue psi5=Torsion(  $a_//n \& Next(a_/5)$  )

### 2.21.80. Covalent neighbors of an atom

Next (as\_bond) - selects atoms forming covalent bonds with the selected single atom.

Example:

```
BS his
display
display a_/his/he2 ball red
display Next( a_/his/he2 bond ) ball magenta # show atom preceding he2
cd2_neigh = Next( a_//cd2 bond )
for i=1,Nof(cd2_neigh)
    nei = cd2_neigh[i]
    print " Distance between a_//cd2 and ",Sum(Name(nei)), " = ", Distance( a_//cd2 nei)
endfor
```

### 2.21.81. Nof

Number-OF-elements function.

Nof ({ iarray | rarray | sarray }) - returns integer number of elements in an array

Nof (*ali\_*) - returns integer number of sequences in the specified alignment *ali\_* (see also Length( *alignment*)).

Nof (*matrix*) – returns integer number of rows in a matrix (see also Length(*matrix*) function which returns number of columns).

Nof (*map*) – returns integer number of grid points in a map.

Nof (grob) – returns integer number of points in graphics object.

Nof ( {  $os_{|} ms_{|} rs_{|} as_{|} vs_{|}$  } ) - returns integer number of selected objects, molecules, residues, atoms or variables respectively.

Nof ( {  $os_{1} | ms_{1} | rs_{2}$  } [ atom ] ) – returns an array of the numbers of atoms in each selected unit. This function automatically excludes the 'virtual' atoms added automatically to the ICM-objects (equivalent to the a\_//!vt\* selection ). Example:

```
read mol s_icmhome+"ex_mol.mol"
Nof( a_*. atom )
```

Nof ( { atoms | residues | molecules | objects | conf | tether | vrestraint } ) – returns the total integer number of atoms, residues, molecules, objects, stack conformations or tethers, variable restraints, respectively. It is safer than the previous command (e.g.  $Nof(a_//*)$ ) since Nof(atoms) will work even if object does not exist.

Nof (library) – returns 1 if the force field parameter library is loaded and 0 otherwise.

Nof (library) - returns 1 if the mmff library is loaded and 0 otherwise.

Nof (site [*ms*]) - returns integer number of sites in the selected molecule or the current object.

Nof ( { *s\_stackFileName* | *s\_objFileName* } ) - returns integer number of conformations in the specified

Example:

```
for i=1,Nof("def.cnf",conf) # stack is NOT loaded
  read conf i
endfor
```

Nof (string, *substring*) – returns integer number of occurrences of *substring* in a *string*. E.g. Nof ("ababab", "ba") returns 2

Nof (string, substring, pattern) – returns integer number of occurrences of regular pattern in a string. E.g. Nof ("ababab", "b?") returns 2

Example with a strange DNA sequence dn1:

```
if(Nof(String(dn1),"[!ACGT]" pattern) > 0.5*Length(dn1)) print " Warning> Bad DNA sequence"
```

Nof ({ iarray | rarray | sarray | sequence | aselection | vselection | alignment | matrix | map | grob | string | object } [ select ]) - returns integer number of ICM-shell variables of specified type. With the select option it returns the number of GUI-selected objects (exceptions: for aselection, vselection, string, object).

Examples:

if ( Nof( sequence select ) == 2 ) a = Align( select )

#### 2.21.82. Norm

normalize numbers.

Norm (*map*) – returns the map linearly transformed into the [0.,1.] range.

# 2.21.83. Obj

object level function.

Obj ( {  $ms_{|}rs_{|}as_{|}$  } ) - selects object(s) related to the specified molecules, residues or atoms, respectively.

Examples:

See also: Atom, Res, Mol.

### 2.21.84. Occupancy

function.

Occupancy ( {  $as_{|}rs_{|}$  } ) – returns rarray of occupancy for the specified selection. If residue selection is given, average residue occupancies are returned.

See also: set occupancy.

Examples:

### 2.21.85. Path

function.

Path (*s\_FullFileName*) – returns header sub- string with the path.

Examples:

sPath=Path("/usr/mitnick/hacker.loot") # returns "/usr/mitnick/"

See also: Name() and Extension() functions which return two other components of the full file name.

Path (preference)

returns the path to the directory in which the user preferences file "icm.ini" is stored. This file is always stored in the s\_icmhome, but a user can save his or her own preferences in the  $s_userDir/config/icm.ini$  file.

Example:

read all Path(preference)+"icm.ini" # restore the settings

```
Path(unix)
```

returns a path to the ICM executable. ICM binary can also be found in the Version(full) string.

### 2.21.86. Pattern

function.

Pattern ( $\{s\_consensus | ali\_\}$ [exact]) - returns sequence pattern string which can be searched in a single sequence with the find pattern command or in a database with the find database pattern=s\\_pattern command. If ICM-consensus string s\\_consensus is provided as an argument, the string is translated into a regular pattern expression (e.g. an expression "R+. ...^D" will be translated to "R[KR]?\{3,6\}[ACGS]D"). If alignment  $ali\_$  is given as an argument, the pattern is either extracted directly from the alignment, option exact, or is converted to consensus first, and only after that translated into a pattern. For example, an alignment position with amino acids A and V will be transformed into pattern [AV] with option exact and into pattern [AFILMPVW] without the option. Additionally, the exact option will retain information about the length of the flanking regions.

Example:

```
read sequence s_icmhome + "zincFing"
group sequence aaa
align aaa
show Pattern("#~???A% ?P") # symbols from consensus string
show Pattern(aaa)
show Pattern(aaa exact)
```

Pattern ( $s\_seqPattern$  prosite) – returns string containing the prosite-formatted sequence pattern. The input string  $s\_seqPattern$  is an ICM sequence pattern.

### 2.21.87. Pi

function (or rather a reserved name).

Pi - returns the real value of Pi(3.14...).

Examples:

print Pi/2. </tt>

### 2.21.88. Potential

function.

Potential (*as\_targets as\_charges*) – returns rarray of Nof(*as\_targets*) real values of electrostatic potentials at *as\_targets* atom centers. Electrostatic potential is calculated from the specified charges *as\_charges* and the precalculated boundary (see also REBEL, make boundary and How to evaluate the pK shift).

Examples:

### 2.21.89. Power

mathematical function.

Power (r\_base, {  $r_exp | i_exp$  }) – returns real  $r_base^{r_exp}$  or  $r_base^{i_exp}$ . Note that  $r_base$  may be negative if the exponent is an integer, otherwise error will be produced.

Power (r\_base, R\_exp) – returns rarray of the r\_base taken to the R\_exp powers.

Power (r\_base, M\_exp) - returns matrix of the r\_base taken to the M\_exp powers.

Example:

```
Power(2., {1. 2. 3.}) # returns {2.,4.,8.}
```

Power (rarray, *r\_Exponent*) – returns rarray of elements taken to the specified power.

Power (matrix, *integer*) – for square matrix returns the source matrix taken to the specified power. If the exponent is negative, the function returns the n-th power of the **inverse** matrix.

Examples:

### 2.21.90. Probability

function.

Probability (*s\_seqPattern*) – returns the real probability of the specified sequence pattern. To get mathematical expectation to find the pattern in a protein of length L, multiply the probability by L–Length(

s\_seqPattern).

Examples:

Probability (i\_minLen, *r\_Score* [, { identity | similarity | comp\_matrix | sort } ]) - returns the real expected probability that a given or higher score (*r\_Score*) might occur between structurally unrelated proteins (i.e. it is essentially the probability of an error). This probability can be used to rank the results of database searches aimed at fold recognition. A better score corresponds to a lower probability for a given alignment. The four types of scores

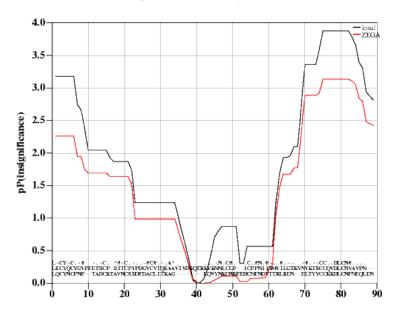
- no argument: alignment score
- identity: the number of identical residues \* 100 and divided by the minimal sequence length
- similarity: alignment score without gap penalties \*100. and divided by the minimal sequence length
- comp\_matrix: alignment score without gap penalties (unnormalized)
- sort: an additional score for ranking

are given in the description of the Score function. Each score has a different distribution which was carefully derived from all-to-all comparisons of sequence of protein domains.

Example: Probability(150, 30, identity)\*55000. is the mathematical expectation of the number of structurally unrelated protein chains of 150 residues with 30% or higher sequence identity which can be found in a search through 55000 sequences. The inverse function is Score.

Probability(ali\_2seq,[i\_windowSize1w\_windowSize2][local])

#### -Log(10., Probability(sx [local]))



- returns the rarray of expected probabilities of local insignificance of the pairwise sequence alignment  $ali_2seq$ . The Karlin and Altschul score probability values (option local) or local ZEGA probabilities (see also Probability(i\_, r\_)) are calculated in multiple windows ranging in size from  $i\_windowSize1$  to  $w\_windowSize2$  (default values 5 and 20 residues, respectively). The exact formulae for the Karlin and Altschul probabilities (option local) are given in the next section and the ZEGA probabilities are given in the Abagyanpaper. The window with the lowest probability value is chosen in each position. The array returned by this function can be used to color-code the regions of insignificant sequence-structure alignment in modeling by homology. One can use the Rarray(R\_,ali\_,seq\_) function to project the array onto selected sequence.

#### Example:

```
read alignment s_icmhome+ "sx" # 2 seq. alignment
read pdb s_icmhome+"x"
p= -Log(Probability(sx))
display ribbon a_
color ribbon a_/A -Rarray(p,sx,cd59)
# Rarray projects the alignment array to the sequence
```

Probability ( $seq_1 seq_2$  [ $i_windowSize1 w_windowSize2$ ]) – returns a dot matrix of probabilities of local statistical comparison between the two sequences. This matrix contains local probability values that two continuous sequence fragments of length ranging from  $i_windowSize1$  to  $w_windowSize2$  have statistically insignificant alignment score, which means that the match is random. Visualization of this matrix allows to see periodic patterns if sequence is compared with itself as well as identity alternative alignments. The formula is taken from the Karlin and Altschul statistics:

P=1-exp(-exp(-Lambda\*Sum(score in window)/K)), where Lambda and K are coefficients depending on the residue comparison matrix.

This example allows to trace the correct alignment despite an about 100 residue insertion:

```
read pdb sequence "2mhb"
read pdb sequence "4mbn"
m=Probability(2mhb_a 4mbn_m 7 30)
print " pProbability: Min=" Min(Min(m)) "Max=" Max(Max(m))
PLOT.rainbowStyle="white/rainbow/red"
# show probability of the chance matching (comparable to the BLAST P-value)
plot area m display color={.2 0.001} transparent={0.2 1.} link grid
# OR show -Log10(Probability)
plot area -Log(m,10) display color={0.7 3.} transparent={0. 0.7} link grid
```

### 2.21.91. Profile

function.

Profile ( alignment ) - creates profile from an alignment

#### 2.21.92. Putenv

function to change or add value to environment.

Putenv (" $s_{environmentName = s_{environmentValue}$ ") - returns a logical yes if the named environment variable is created or modified.

Example:

See also: Existenv(), Getenv().

### 2.21.93. Radius

atomic radii (van der Waals, surface energy, and electrostatic).

Radius (as\_) – returns the real array of van der Waals radii of atoms in the selection.

These radii are used in the construction of the molecular surface (skin) and can be found (and possibly redefined) in the icm.vwt file.

Radius (as\_surface) – returns the rarray of the 'hydration' atomic radii.

These radii are used in construction of the solvent-accessible (surface) and can be found (and possibly redefined) in the icm.hdt file.

Radius (as\_charge) – returns the rarray of the 'electrostatic' atomic radii.

These radii are used for building the skin (analytical molecular surface) for electrostatic dielectric boundary calculation with electroMethod = "boundary element". These parameters can be found (and possibly redefined) in the icm.vwt file.

### 2.21.94. Random

evenly distributed random function.

Random () – returns a pseudo-random real in the range from 0. to 1.

Random (*i\_max*) – returns a pseudo-random integer distributed in [1, *i\_max*]

Random (*i\_min*, *i\_max*) – returns a pseudo-random integer distributed in [*i\_min*, *i\_max*]

Random (*r\_min*, *r\_max*) – returns a pseudo-random real evenly distributed in [*r\_min*, *r\_max*]

Random ( $r_{min}$ ,  $r_{max}$ ,  $i_n$ ) – returns a rarray [1:  $i_n$ ] with pseudo-random real values distributed in [ $r_{min}$ ,  $r_{max}$ ]

Random (*r\_mean*, *r\_std*, *i\_n*, "gauss") – returns a rarray of *i\_n* elements with normally distributed pseudo-random values. The mean and standard deviation are provided as the first two arguments

Random (*r\_min*, *r\_max*, *i\_nRows*, *i\_nColumns*) - returns a matrix [1: *i\_nRows*, 1: *i\_nColumns*] with pseudo-random real values distributed in [*r\_min*, *r\_max*]

Random (i\_nRows, i\_nColumns, r\_min, r\_max) - returns a matrix [1: i\_nRows, 1: i\_nColumns] with pseudo-random real values distributed in [r\_min, r\_max]

Random (sequence) - returns a randomized sequence with the same amino-acid composition

Examples:

```
print Random(5)  # one of the following: 1 2 3 4 or 5
print Random(2,5)  # one of the following: 2 3 4 or 5
print Random(2.,5.)  # random real in [2.,5.]
randVec=Random(-1.,1.,3)  # random 3-vector with components in [-1. 1.]
randVec=Random(3,-1.,1.)  # the same as the previous command
randMat=Random(-1.,1.,3,3)  # random 3x3 matrix with components in [-1. 1.]
randMat=Random(3,3,-1.,1.)  # the same as the previous command
Random(0., 1., 10, "gauss" # normal distribution
a=Random(seq_crn)  # a contains a randomized crambin sequence
```

### 2.21.95. Rarray

real-array function.

Rarray (*i\_NofElements*) – returns a rarray; creates zero-initialized rarray [1: *i\_NofElements*]. You can also create an zero-size real array: Rarray(0).

Rarray (i\_NofElements,  $r_Value$ ) – returns a rarray [1: *i\_NofElements*] with all elements set to  $r_Value$ .

Rarray (i\_NofElements, r\_From,  $r_To$ ) – returns a rarray [1: *i\_NofElements*] with elements ranging from  $r_From$  to  $r_To$ .

Rarray (*iarray*) – converts *iarray* into a rarray.

Rarray (sarray) - converts sarray into a rarray.

Examples:

```
      a=Rarray(54)
      # create 54-th dimensional vector of zeros

      a=Rarray(3,-1.)
      # create vector {-1.,-1.,-1.}

      a=Rarray(5,1.,3.)
      # create vector {1., 1.5, 2., 2.5, 3.}

      a=Rarray({1 2 3})
      # create vector {1. 2. 3.}

      a=Rarray({"1.5" "2" "-3.91"})
      # create vector {1.5, 2., -3.91}
```

#### rarray sequence projection

Rarray ( R\_ali ali\_from { seq\_ | i\_seqNumber } )

- returns a projected rarray. The  $R_{ali}$  rarray contains values defined for each position of alignment *ali\_from*. The function squeezes out the values which correspond to insertions into sequence *seq\_*, that is, in effect, projects the alignment array  $R_{ali}$  onto sequence *seq\_*.

#### Projecting from one sequence to another sequence via alignment.

Let us imagine that we have two sequences, *seq1* and *seq2* which take part in multiple sequence alignment *ali*. The transfer of property R1 from *seq1* to *seq2* can be achieved via *two transfers* :

1. *seq1* to *ali* : RA = Rarray( R1 seq1 ali r\_gapValues ) 2. *ali* to *seq2* : R2 = Rarray( RA ali seq2 )

Now R2 has the same dimension as seq2. Values aligned with  $\sim$  seq1 are transferred by alignment, other values are set to  $r_gapValues$ .

See also:

```
String(s_,R_ali_seq_)function to project stringsRarray(R_seq seq_ali_to r_gapDefault)function to project from sequence to alignmentProbabilityfunctionRarray(rarray reverse)function
```

- converts input real array into an rarray with the reversed order of elements. Example:

```
Rarray({1. 2. 3.} reverse) # returns {3. 2. 1.}
```

#### Transfer real sequence properties by alignment

Rarray ( R\_seq { seq\_| i\_seqNumber } ali\_to r\_gapDefault )

- projects the input rarray from  $seq_$  to  $ali_{to}$  (the previous function does it in the opposite direction). The  $R_{seq}$  rarray contains values defined for each position of the sequence  $seq_{-}$ . The function fills the gap positions in the output array with the  $r_{gap}Default$  values.

Combination of this and the previous functions allow you to project any numerical property of one sequence to another by projecting the r1 property of seq1 first to the alignment and than back to seq2 (e.g. Rarray(Rarray(r1,seq1,a,99.), a, seq2)).

This function can also be used to determine alignment index corresponding to a sequence index. Example:

See also the String(*s\_,R\_,seq\_,ali\_,s\_defChar*) function to project strings.

#### Assign arbitrary amino-acid property to a sequence

Rarray (sequence R\_26resProperty)

- returns a rarray of residue properties as defined by *R\_26resProperty* for 26 residue types (all characters of the alphabet) and assigned according to the amino-acid *sequence*.

Example with a hydrophobicity property vector:

### 2.21.96. Real function

generally converts things to a real.

Real (*integer*) – converts integer to real number.

Real ( *iarray* ) see Rarray( *iarray*).

Real (*string*) – converts string to real number. The conversion routine ignores trailing non–numerical characters.

Examples:

```
s = "5.3"
a = Real(s)  # a = 5.3
s = "5.3abc"  # will ignore 'abc'
a = Real(s)  # the same, a = 5.3
```

Real (*rarray*) – returns the first element of the real array. With that trick one can also transform real array with one element into the real number. You may also convert a one element array into a real with the Sum or the Mean functions. If there are more then one elements, the first element is taken. Important for assignments.

Example:

 $a[2,3]=Real(Value(v_/2/phi)) \# a is a matrix, a[2,3] expects a real$ 

#### 2.21.97. Remainder function.

Returns the remainder; similar to, but different from the Mod function.

functiondescriptionexampleRemainder(x,y)brings x to [-y/2, y/2] rangeRemainder(17.,10.)=>-3.Mod(x,y)brings x to [0, y] rangeMod(17.,10.)=>7.Remainder (i\_divisor,  $[i_divider]$ ) - returns the integer

Remainder (r\_divisor, [ $r_divider$ ]) – returns the real remainder r = x – n\*y where n is the integer nearest the exact value of x/y; if | n-x/y=0.5 then n is even. r belongs to [ $-\frac{y}{2}$ ,  $\frac{y}{2}$ ] range

Remainder (iarray, [*i\_divider*]) – returns the iarray of remainders (see the previous definition).

Remainder (rarray, [r\_divider]) – returns the rarray of remainders.

The default divider is 360. (real) or 360 (integer) since we mostly deal with angles.

Examples:

```
# transform angle to the standard
# [-180., 180.] range. (Period=360 is implied)
phi=Remainder(phi)
# we assume that you have two objects
# with different conf. of the same molecule
phiPsiVec1 = Value(v_1.//phi,psi)
phiPsiVec2 = Value(v_2.//phi,psi)
# average angular
# deviation
```

### 2.21.98. Replace

- text substitution function.

Replace (*s\_source s\_regularExpression s\_replacement*) – returns a string, which is a copy of the source *string* with globally substituted substrings matching *s\_regularExpression* by the replacement string *s\_replacement*.

Example:

a=Replace(" 1crn "," ","") # remove empty space

Replace ( $s_source S_from Array S_to Array$ ) – make several replacements in a row. The size of the two arrays must be the same.

Example which generates a complimentary DNA strand (actually there is a special function Sequence( *seq\_*, reverse) which does it properly).

```
invertedSeq = String(0,1,"GTAAAGGGGTTTTCC") # result: CCTTT..
complSeq=Replace(invertedSeq,{"A","C","G","T"},{"T","G","C","A"})
# result: GGAAA...
```

Replace (*s\_source S\_fromArray s\_replacement*) – replace several strings by a single other string. If *s\_replacement* is empty, the found substrings will be deleted.

Example which generates a complimentary DNA strand:

```
cleanStr=Replace("XXTEXTYYTEXT",{"XX","YY"},"")
```

Replace (*S\_s\_regularExpression s\_replacement*) – returns a sarray with globally substituted elements (the original sarray remains intact).

Examples:

aa={"Terra" "Tera" "Tera" "Ttera"}
show column aa Replace(aa "er?" "ERR") Replace(aa "\*[tT]" "Shm")

Replace (*S\_S\_fromArray S\_toArray*) – returns a sarray with multiple substitutions.

Replace ( $S_s$ -regularExpression s\_replacement) – returns a sarray with multiple substitutions to a single string.

### 2.21.99. Res

residue selection function.

```
Res ( { os_{1} | ms_{1} | as_{2} } ) – selects residue(s) related to the specified objects ( os_{1}), molecules ( ms_{1}) or atoms ( as_{1}), respectively.
```

Examples:

```
show Res( Sphere(a_1/1/* 4.) ) # show residues within 4 A
# vicinity from the firsts one
```

See also: Atom(), Mol(), Obj().

Res (cursor) – returns residue at which interactive residue cursor is set. See also displaycursor{display cursor}, color cursor.

# 2.21.100. Res(ali ..): from sequence positions in subalignment to residue selection

Res (*ali\_* { *seq\_* | *i\_sequence* }) – returns residue selection corresponding to the aligned positions of the specified sequence. The sequence can be specified by its order number in the alignment (e.g. lcrn\_m in the example below has number 1), or by name.

An example in which we find residue selection corresponding to the aligned part of crambin sequence lcrn\_m:

```
read pdb "lcrn"
read alignment "bb" # a short sublalignment of 1crn m
link a_*. bb
              # link objects and sequences
show bb
#>ali bb
# Consensus .C CP~.#A.^.#
            TC-CPSIVARSNF-----
1crn_m
            PCGCPDGIAIARIYPFAVG
а
            EE E HHHHHHH
#1crn_m
# 1crn m a nID 4 Lmin 46 ID 8.7 % Score 4.75 Sim 15.54 Gap 2.40 nOverlap 12 pP 0.16
#MATGAP gonnet 2.4 0.15
Res(bb1)
- Num Res. Type ---- SS Molecule ---- Object - sf - sfRatio
   2 thr
          Amino
                    T E m
                                    lcrn
                                            0.0 0.00
   3 cys
           Amino
                   C E m
                                             0.0 0.00
                                    lcrn
   4 cys
                   C E m
          Amino
                                             0.0 0.00
                                    1crn
                   P____m
S____m
IH___m
           Amino
   5 pro
                                    lcrn
                                             0.0
                                                 0.00
   6
     ser
           Amino
                                    1crn
                                             0.0
                                                 0.00
   7
     ile
                                    1crn
                                                 0.00
           Amino
                                             0.0
   8 val
           Amino
                    VΗ
                        т
                                    1crn
                                             0.0
                                                 0.00
   9
     ala
           Amino
                   ΑH
                                    1crn
                                            0.0
                                                 0.00
                        m
  10 arg
           Amino
                   R H
                                    1crn
                                            0.0
                                                 0.00
                        m
                                    1crn
  11 ser
           Amino
                   S H
                        т
                                            0.0
                                                 0.00
          Amino
                   NH m
                                    1crn
                                            0.0 0.00
  12 asn
  13 phe Amino
                  FHm
                                            0.0 0.00
                                    1crn
```

### 2.21.101. Resolution

- returns the X-ray resolution in Angstroms.

Resolution () - returns the real resolution of the current object.

Resolution (*os\_object*) – returns the real X-ray resolution for the specified object. The resolution is taken from the PDB files.

Examples:

```
res=Resolution(a_lcrn.) </tt>
print "PDB structure lcrn: resolution = ", res, " A"
```

Resolution (*s\_pdbFileName* pdb) – returns the real resolution of the specified pdb-file. The function returns 9.90 if resolution is not found.

Resolution ( $T_factors [R_6cell]$ ) – returns the rarray of X-ray resolution for each reflection of the specified structure factor table. The resolution is calculated from h, k, l and cell parameters taken from  $R_6cell$  or the standard defCell shell rarray.

Example:

```
read factor "igd" # read h,k,l,fo table from a file
read pdb "ligd" # cell is defined there
defCell = Cell(a_) # extract the cell parameters from the object
group table append igd Resolution(igd) "res"
show igd
```

### 2.21.102. Rfactor

crystallographic R-factor.

```
Rfactor (T_factors) – returns the real R-factor residual calculated from the factor-table elements T_factors.fo and T_factors.fc. Reflections marked with T_factors.free = 1 are ignored.
```

### 2.21.103. Rfree

crystallographic free R-factor.

Rfree ( $T_factors$ ) – returns the real R-factor residual calculated from the factor-table elements  $T_factors.fo$  and  $T_factors.fc$ . Only reflections marked with  $T_factors.free = 1$  will be used.

### 2.21.104. Rmsd

Root-Mean-Square-Deviation function.

Rmsd ( { iarray | rarray | matrix | map } )

- returns the real standard deviation (sigma) from the mean for specified sets of numbers

Rmsd(as\_select1 as\_select2[{ { ali\_lalign } | exact }])

- returns the real root-mean-square-distance between two aligned sets after these two sets are optimally superimposed using McLachlan's algorithm.

The optional third argument defines how atom-atom alignment is established between two selections (which can actually be of any level atom selection  $as_{,}$  residue selection  $rs_{,}$  molecular selection  $ms_{,}$ , or object selection  $os_{,}$  see alignment options). Number of equivalent atom pairs is saved in i\_out. Two output selections as\_out and as2\_out contain corresponding sets of equivalent atoms.

See also: superimpose and Srmsd ().

Examples:

read pdb "1mbn"	# load myoglobin
read pdb "lpbx"	# load alpha and beta
	<pre># subunits of hemoglobin</pre>
print Rmsd(a_1.1 a_2.1 align)	# myo- versus alpha subunit
	# of hemo- all atoms
print Rmsd(a_1.1//ca a_2.1//ca align)	# myo- versus alpha subunit
	# of hemo- Ca-atoms

print  $Rmsd(a_1./4, 29/ca a_2.1/2, 102/cb exact) # exact match$ 

### 2.21.105. Rot

rotation matrix function.

Rot ( $R_12$  transform Vector) – extracts the 3x3 rotation matrix from the transformation vector.

Rot ( $R_axis, r_Angle$ ) – returns matrix of rotation around 3-dimensional real vector  $R_axis$  by angle  $r_Angle$ .

Examples:

```
# rotate molecule by 30 deg. around z-axis rotate a_* Rot({0. 0. 1.}, 30.)
```

Rot ( R\_3pivotPoint R\_3axis, r\_Angle )

- returns rarray of transformation vector of rotation around 3-dimensional real vector  $R_axis$  by angle  $r_Angle$  so that the pivotal point with coordinates  $R_3pivotPoint$  remains static.

Examples:

```
# rotate by 30 deg. around {0.,1.,0.} axis through the center of mass
nice "lcrn"
R_pivot = Mean(Xyz(a_//*))
transform a_* Rot(R_pivot, {0. 1. 0.}, 30.)
```

### 2.21.106. Sarray

sarray function.

Sarray(integer)

- returns empty sarray of specified dimension

Sarray (*integer s\_Value*) – returns sarray of specified dimension initialized with *s\_Value* 

Sarray (*string*) – converts the input string into a ONE-dimensional sarray. To split a string into individual lines, or to split a string into a sarray of characters, use the Split() function.

Sarray (*iarray*) – converts input iarray into an sarray

Sarray (rarray) - converts input rarray into an sarray

Sarray (*rs*) – converts input residue selection into an sarray of residue ranges, e.g.: {"a\_a.b/2:5", "a\_a.b/10:15",..} . See also String(*rs*) which returns one string. Multiple selections

Sarray (stack, *vs\_var*) – creates a string representation of all the conformations in the stack Variable selection allows to choose the conformational feature you want. Character code:

Backbone (phi,psi pairs):

• 'B': -200 < phi < -80, 140 < psi < 200• 'A': -101 < phi < -24, -81 < psi < 4• 'g': -169 < phi < -15, -64 < psi < 54• 'd': -211 < phi < -5, 8 < psi < 136• 'L': 24 < phi < 101, -4 < psi < 81• '\_': the rest

Sidechain (chi1):

- 'M': -120 <= xi1 < 0 • 'P': 0 <= xi1 < 120
- 'T': 120 <= xi1 < 240

Example:

```
show Sarray(stack,v_/2:10/x*)  # coding of side-chain conformations
show Sarray(stack,v_//phi,psi)  # backbone conformation character coding
show Sarray(stack,v_/2:10/phi,PSI)  # character coding of a chain fragment
```

(Note use of special PSI torsion in the last example.)

Other examples:

ss=Sarray(5) # create empty sarray of 5 elements ss[2]="thoughts" # assign string to the second element of the sarray sa=Sarray("the first element")
show Sarray(Count(1 100)) # string array of numbers from 1 to 100

Sarray (sarray reverse)

- converts input sarray into an sarray with the reversed order of elements. Example:

Sarray({"one","two"} reverse) # returns {"two","one"}

See also:  $Iarray(I_reverse)$ ,  $Rarray(S_reverse)$ , String(0,1,s)

### 2.21.107. Score

function.

Score ( $R_1, R_2$ ) – returns the real shift between two distributions based on the overlap between the two real arrays. This measure of changes between -1 and 1.. (all values of  $R_1$  are smaller than all values of  $R_2$ ) and +1. (all values of  $R_1$  are greater than all values of  $R_2$ ) and may serve as a ranking criterion.

Examples:

Score ( sequence1, sequence2 )

- returns the real score of the Needleman and Wunsch alignment.

Each pair of aligned residues contributes according to the current residue comparison table, which is normalized so that the average diagonal element is 1. Insertions and deletions reduce the score according to the gapOpen and gapExtension parameters. Approximately, the score is equal to the number of residue identities.

Examples:

```
read sequence "seqs"
a = Score( Azur_Alcde Azur_Alcfa ) # it is around 90.
```

See also: Distance().

Score (  $ali_{, [{ dentity | similarity | comp_matrix | sort }]) - returns the real score of the given alignment calculated by different methods:$ 

• *no second argument* : the straight Needleman and Wunsch score: aligned residues score according to the residue comparison table, gaps according to the gapOpen and gapExtension parameters.

- identity the number of identical residues in the alignment divided by the smallest sequence length and multiplied by 100 %.
- comp\_matrix the alignment score without the gap component. It contains only the total score of the aligned residues calculated from the residue comparison table and does not include penalty term.
- similarity the alignment score without the gap component multiplied by 100. and divided by the smallest sequence length.
- sort a score occasionally used for ranking/sorting the alignments in fold recognition. Currently it is equal to the **comp\_matrix\_score** 1.3\***totalGapPenalty**

Score (i\_minLen,  $r_Probability$  [, { identity | similarity | sort } ]) – returns the real threshold score at a given  $r_Probability$  level of occurrence of alignment with a protein of unrelated fold. The threshold is related to the corresponding method of the score calculation (see above). For example,

```
Score( 150, 1./55000.,identity)
```

gives you the sequence identity percentage for sequences of 150 residues at which only one false positive is expected in a search through the Swissprot database of 55000 sequences.

See also the inverse function: Probability.

### 2.21.108. Select

Selection of atoms according to their coordinates or properties.

Atoms with certain Cartesian coordinates can also be selected by multiplying selection to a box specified by 6 real numbers  $\{x,y,z,X,Y,Z\}$ , e.g. show  $a_{//*} \{-1.,10.,2.,25.,30.,22.\}$  or  $a_{//*}$  Box().

See also: display box and the Box function.

Select (*as\_s\_condition* [*r\_Value*]) – returns a subselection of atom selection *as\_* according to the specified condition *s\_condition*.

Three example conditions:

"X >= 2.0", "Bfactor != 25.", "charge == 0.". Allowed properties and their aliases (case does not matter, the first character is sufficient) are as follows:

- x,y,z atomic coordinates ("x","y","z")
- bfactor ("bfactor","b","B")
- occupancy ("o")
- charge ("charge", "c", "q")
- accessible surface area ("area", "a")
- user-field ("u") which can be set with set field and extracted with the Field function.
- residue user-fields: "u","v","w" for the 1st, 2nd and 3rd field, respectively.

Note: do not forget to calculate surface in advance with the show area command. Allowed comparisons: (==!=><>=<=). The value can either be specified inside the string or used as a separate argument

r\_Value.

See also the related functions: Area, Bfactor, Xyz, Charge, Field.

Examples:

```
build string "se glu arg" show Select(a_//* "charge < 0.")|Select(a_//c* "x> -2.4") show Select(a_//c* , "x>", -2.4) show Select(a_/* , "w>3.") # 3rd res. user field greater than 3.
```

Select ( as\_sourceSelection os\_targetObject )

- returns the source selection *as\_sourceSelection* from a source object which is transferred to another object (*os\_targetObject*). The two objects must be identical in content. Example:

### 2.21.109. Sequence

function.

Sequence (*as\_select*) – returns sequence extracted from specified residues.

Sequence (*string* [nucleotide | protein]) - converts a string (e.g. "ASDFTREW") into an ICM sequence object. By default the type is "protein". To reset the type use the set type *seq* { nucleotide | protein } command.

Examples:

Sequence  $(ali_)$  – returns a chimeric sequence which represents the strongest character in every alignment position.

Sequence (*prf\_*) – returns a chimeric sequence which represents the strongest character in every profile position.

### 2.21.110. reverse complement dna sequence function

Sequence (seq\_DNAsequence reverse) - returns the reverse complement DNA sequence:

nucleotide	complement
A = Adenosine C = Cytidine	T (replace by U for RNA)
G = Guanosine	G
T = Thymidine	A
U = Uridine	A
R = puRine (G A)	Y
Y = pYrimidine(T C)	R
K = Keto (G T)	M
M = aMino (A C)	K
S = Strong (G C)	S
W = Weak (A T)	W
B = !A (GTC)	V
D = !C (G A T)	Н
H = !G (A C T)	D
V = !T (G C A)	В
N = aNy	N

### 2.21.111. Sign

transfer-of-sign function. It returns the value (or values) of sign { -1./0./+1.} of its argument.

Sign (*real*) – returns real sign of the argument.

Sign(integer) - returns integer.

Sign(iarray) - returns iarray.

Sign(rarray) - returns rarray.

Examples:

```
Sign(-23)
-1
Sign(-23.3)
-1.
Sign({-23,13})
{-1,1}
Sign({-23.0,13.1})
{-1.,1.}
```

### 2.21.112. Sin

sine trigonometric function. Arguments are assumed to be in degrees.

Sin ({ real | integer }) - returns the real sine of its real or integer argument.

Sin (rarray) – returns the rarray of sines of rarray elements.

Examples:

### 2.21.113. Sinh

hyperbolic sine function.

Sinh ( { real | integer } ) – returns the real hyperbolic sine of its real or integer argument. Sinh(x)=0.5(  $e^{iz} - e^{-iz}$ )

Sinh (rarray) – returns the rarray of hyperbolic sines of rarray elements.

Examples:

```
print Sinh(1.)  # equal to 1.175201
print Sinh(1)  # the same
print Sinh({-1., 0., 1.})  # returns {-1.175201, 0., 1.175201}
```

# 2.21.114. Site

site selection function

Site (*s\_siteID* [*ms\_*]) – returns the iarray of the site numbers in the selected molecule. The default is all the molecules of the current object.

Example:

```
nice "lest" # contains some sites
delete site a_1 Site("CONFLICT",a_1)
```

### 2.21.115. Smiles

convert chemical structure into a Smiles string.

Smiles (*as*\_) – returns the smiles-string with the coded representation of the chemical structure of selected fragment.

See also: build smiles, String( *as\_*) - chemical formula.

### 2.21.116. Smooth

sliding window averaging, convolution, 3D-gaussian smoothing, map smoothing and function derivatives.

#### Smooth

Smooth (R\_source, [ $i\_windowSize$ ]) – returns the window-averaged rarray. The array is of the same dimension as the *R\_source* and  $i\_windowSize$  is set to windowSize by default. An average value is assigned to the middle element of the window.  $i\_windowSize$  must be an odd number. At the array boundaries the number of averaged elements is gradually reduced to one element, i.e. if  $i\_windowSize=5$ , the 3rd element of the smoothed array will get the mean of R1,R2,R3,R4,R5, the second element will get the mean of R1,R2 and R3, and the first element will be set to R1.

Smooth (R\_source,  $R_weightArray$ ) – returns the rarray of the same dimension as the  $R_source$ , performs convolution of these two arrays. If  $R_weightArray$  contains equal numbers of  $1./i_windowSize$ , it is equivalent to the previous option. For averaging, elements of  $R_weightArray$  are automatically normalized so that the sum of all elements in the window is 1.0.

Normalization is not applied if the sum of elements in the *R\_weightArray* is zero. Convolution with such an array may help you to get the derivatives of the *R\_source* array. Use:

```
{-1.,1.}/Xstep  # for the first derivative
{1.,-2.,1.}/(Xstep*Xstep) # for the second derivative
{-1.,3.,-3.,1.}/(Xstep*Xstep*Xstep) # for the third derivative
# ... etc.
```

Examples:

```
gauss=Exp( -Power(Rarray(31,-1.,1.) , 2) ) # N(0.,1.) distribution on a grid
x = Rarray(361,-180.,180.) # x-array grows from 0. to 180.
a = Sin(x) + Random(-0.1,0.1,361) # noisy sine
b = Smooth(a,gauss) # gauss averaging
# see how noise and smooth signals look
plot x//x a//b display {-180.,180.,30.,10.}
# take the first derivative of Sin(x)
c = Smooth(Sin(x), {-1., 1.}) * 180.0 / Pi
# plot the derivative
plot x c display {"X", "d(Sin(X))/dX", "Derivative"}
```

#### Smooth: three-dimensional averaging of residue properties

Smooth (rs\_, R\_property,  $r\_smoothRadius$ ) – gaussian averaging of property array  $R\_property$  of residues  $rs\_$ . The averaging is performed according to the spatial distance between residue Ca atoms. The function returns the rarray of the residue property AVERAGED in 3D using spherical gaussian with sigma of  $r\_smoothRadius$ . Each residue contributes to the smoothed property with the weight of  $exp(-Dist\_i_j^2/r\_smoothRadius^2)$ .

The interresidue distances  $Dist_i_j$  are calculated between atoms carrying the residue label (normally  $a_//ca$ ). These atoms can be changed with the set label command. Array  $R_1$  is normalized so that the mean value is not changed.

The distances are calculated between

Examples:

```
nice "ltet" # it is a macro displaying ribbon++
R = Bfactor(a_/A) # an array we will be 3D-averaging
color ribbon a_/A Smooth(a_/A R 1.)//5.//30. # averaging with 1A radius
color ribbon a_/A Smooth(a_/A R 5.)//5.//30. # with 5A radius
color ribbon a_/A Smooth(a_/A R 10.)//5.//30. # with 10A radius
# 5.//30. are appended for color scaling from 5. (blue) to 30.(red)
# rather than automated rescaling to the current range
```

set field a\_/A Smooth(a\_/A R 5.) show Select( a\_/A "u>30." ) # select residues with 1st field > 30.

#### Smooth: expanding alignment gaps

Smooth (ali\_, [*i\_gapExpansionSize*]) – returns a transformation of the initial alignment in which every gap is widened by the *i\_gapExpansionSize* residues. This transformation is useful in modeling by homology since the residue pairs flanking gaps usually deviate from the template positions.

The default *i\_gapExpansionSize* is 1 (the gaps are expanded by one residue)

#### Smooth: transforming three-dimensional map functions.

Smooth(map\_,["expand"])

#### weighted 3D-window averaging

Smooth  $(map_)$  – returns map with averaged map function values. By default the value in each grid node is averaged with the six immediate neighbors (analogous to one-dimensional averaging by Smooth(R,{1,2,.1}). By applying Smooth several times you may effectively increase the window. This operation may be applied to "ge", "gb", "gs" and electron density maps

#### low-values propagation

Smooth (*map*\_"expand") – returns map in which the low values were propagated in three dimensions to the neighboring nodes. This trick allows to generate more permission van der Waals maps.

This operation may be applied to "gh", "gc" and electron density maps.

Examples:

m\_gc = Smooth(Smooth(m\_gc "expand"), "expand" )

See also: map .

# 2.21.117. Sql

functions to connect to a MySql server and run SQL queries. This function has the following properties:

- it supports one connection at a time
- requires a running MySql server (see www.mysql.com) with a database
- the record columns from the database are converted into the ICM sarrays, rarrays and iarrays with one exception. The mysql BIGINT values can not be converted into iarray and are stored as a

string array (sarray).

Sql (connect *s\_host s\_loginName s\_password s\_dbName*) – returns the logical status of connection to the specified server. The arguments are the following:

- *s\_host* (default "localhost") the host name
- *s\_loginName* (default "root")
- *s\_password* the database password
- *s*\_*dbName* the database name

Sql (*s\_SQLquery*) – returns the table of the selected records. Some SQL commands are not really queries and do not return records, but rather perform certain operations (e.g. insert or update records, shows statistics). In this case an empty table is returned. An example:

```
if !Sql( connect "localhost","john","secret","swiss") print "Error"
id=24
T =Sql( "SELECT * FROM swissprothits WHERE featureid="+id )
sort T.featureid
web T
# Another example
tusers = Sql("select * from user where User=" + s_usrName )
Sql(off)
```

Sql (off) – disconnects from the database server and returns the logical status.

### 2.21.118. Sqrt

square root function.

Sqrt (*real*) – returns the real square root of its real argument

Sqrt (*rarray*) – returns rarray of square roots of the *rarray* elements.

Sqrt (*matrix*) – returns matrix of square roots of the *matrix* elements.

Examples:

```
show Sqrt(4.) # 2.
show Sqrt({4. 6.25}) # {2. 2.5}
```

# 2.21.119. Sphere

sphere selection function. It returns a selection containing atoms, residues or molecules within a certain radius around the initial selection. It returns atom selection which can be then converted into residue and molecules with the Res and Mol functions respectively. The default value is defined by the selectSphereRadius . ICM-shell variable which is equal to 5.0 A by default.

Sphere ( {  $as\_source | g_ | R_xyz | M_xyz$  } [  $as\_whereToSelect$ ] [{  $i\_Radius | r\_Radius$ }]) this function always returns a selection of **atoms** in a certain vicinity of the following:

• a group of atoms ( *as\_*)

- any vertex point of a grob ( g\_)
- a point in space (*R\_xyz*)
- a group of points in space  $(M_xyz)$  (e.g. see the Xyz function)

The atoms will be searched in the specified selection *as\_whereToSelect* if the second selection is explicitly specified. If only one atom selection is specified, the atoms will be selected from the same object.

Use the selection level functions (Res, Mol, and Obj to convert the atom selection into residues, molecules or objects, respectively (e.g. Res(Sphere( $a_{115}, 4.)$ )). For example, selection

```
show Sphere( a_subA/14:15/ca,c,n,o , 5.2)
Res(Sphere( a_1.2 a_2.)) # residues of a_2. around ligand a_1.2
```

# 2.21.120. Split

function.

Split (s\_multiFieldString, [s\_Separators]) – returns sarray of fields separated by s\_Separators. By default s\_Separators is set to s\_fieldDelimiter. Multiple spaces are treated as one space, while all other multiple separators lead to empty fields between them. If s\_Separator is an empty string (""), the line will be split into individual characters. To split a multi–line string into individual lines, use Split(s\_, "\n").

Examples:

See also: Field().

### 2.21.121. Srmsd

"static" root-mean-square deviation function.

Srmsd (*as\_select1 as\_select2* [ { { align | *ali\_* } | exact } ]) - returns real value of root-mean-square deviation. Similar to function Rmsd, but works without optimal superposition, i.e. atomic coordinates are compared as they are without modification. Number of equivalent atom pairs is saved in i\_out (see alignment options).

Examples:

superimpose a_1.1 a_2.1	# two similar objects, each
	<pre># containing two molecules</pre>
print Srmsd(a_1.2//ca a_2.2//ca)	<pre># compare how second molecule</pre>
	# deviates if first superimposed

Srmsd (*as\_select*) – returns real root-mean-square length of absolute distance restraints (tethers) for the tethered ICM-objects.

Equivalent to

```
Sqrt(Energy("tz")/Nof(tether)) after show energy "tz" .
```

### 2.21.122. String

function.

String (sequence) - converts sequence into a string

String (integer) - converts integer into a string

String (*real* [*i\_nOfDecimals*]) – converts real into a string. It also allows to round a real number to a given number of digits after decimal point.

String (string, i\_NofRepeats) - repeat specified string i\_NofRepeats times

String (string, all) – adds flanking quotes and extra escape symbols to write this string in a form interpretable in shell in \$string expression.

String (s\_input, s\_default) – if the input s\_input string is empty returns the s\_default, otherwise returns the s\_input string

String (string, i\_offset, *i\_length*) – returns substring of length *i\_length*. If *i\_length* is negative returns substring from the offset to the end.

String (*sarray*) – extracts the first string from the array

String ({ *iarray* | *matrix* } } [ *s\_translateString* ]) – converts numbers into a string or ascii characters (the "Ascii art", i.e. 12345 –> "..:\*#").

The range between the minimal and maximal values is equally divided into equal subranges for each character in the string. This function is useful for ascii visualization of arrays and matrices. The default translation string is ".:\*0#". Another popular choice is "0123456789".

Examples:

```
file=s_tempDir//String(Energy("ener"))  # tricky file name
show Index(String(seq), "AGST")  # use Index to find seq. pattern
tenX = String("X",10)  # generate "XXXXXXXXX"
read matrix
show String(def," ..:*#")
```

See also: show map.

String (i\_from, i\_to, string) – returns substring starting from  $i_from$  and ending at ~i\_to. If  $i_from$  is less than  $i_to$  the string is **inverted**. Zero value is automatically replaced by the string length, -1 is the last but one element etc.

Examples:

```
String(1,3,"12345") # returns substring "123"
String(4,2,"12345") # returns substring "432"
String(1,0,"12345") # returns "12345"
String(0,1,"12345") # returns INVERTED string "54321"
String(-1,1,"12345") # returns "4321"
```

 $String(ali_)$  - converts the alignment into a multiline string. You can further split it into individual lines like "--NSGDG" with the Split( $String(ali_)$ ) command. The offset in a specific sequence and its number can be found as follows.

Examples:

String (*ali*\_tree) - returns a Newick tree string describing the topology of the evolutionary tree. The format is described at http://evolution.genetics.washington.edu/phylip/newicktree.html.

#### Example:

```
read alignment s_icmhome+"sh3"
show String(sh3 tree)
```

#### Projecting properties from alignment to a member sequence.

String ( s\_ali ali\_from { seq\_| i\_seqNumber } )

- returns a projected string. The  $s_{ali}$  string contains characters defined for each position of alignment *ali\_from*. The function squeezes out the characters which correspond to insertions into sequence  $seq_{-}$ . This operation, in effect, projects the alignment string  $s_{ali}$  onto sequence  $seq_{-}$ .

See also the Rarray(R\_,ali\_,seq\_) function to project rarrays.

Example (projection of the consensus string onto a sequence):

```
read alignment s_icmhome+"sh3" # 3 seq.
cc = Consensus(sh3)
show String(Spec)//String(cc,sh3,Spec)
```

#### Projecting properties from member sequence to alignment

String ( $s\_seq$  {  $seq\_i\_seqNumber$  }  $ali\_to s\_gapDefChar$ ) – projects the input string from  $seq\_to$   $ali\_to$  (the previous function does it in the opposite direction). The  $R\_seq$  string contains characters defined for each position of the sequence  $seq\_$ . The function fills the gap positions in the output with the  $r\_gapDefChar$  character. Combination of this and the previous functions allow you to project any string s1 from one sequence to another by projecting the s1 of seq1 first to the alignment and than back to seq2 (e.g.

String(String(s1,seq1,a,"X"), a, seq2)).

See also the Rarray( *R\_seq\_ali\_r\_gapDefault* ) function to project real arrays.

Example (transfer of the secondary structure from one sequence to another):

```
read alignment s_icmhome+"sh3" # 3 seq.
ssFyn = Sstructure(Fyn)
set sstructure Spec String(String(ssFyn,Fyn,sh3,"_"),sh3,Spec)
show Spec
```

#### String( selection ): converting selections into the text form

String ( {  $os_{|}ms_{|}rs_{|}as_{|}$  } [  $i_{number}$  ] ) converts a selection into a compact string form. Continuous blocks of selected elements in different molecules or objects are separated by vertical bar ( | ) which means logical or (e.g. a\_a.1:4 | a\_b.2,14 ) You can also divide this selection info a string array with the Split function.

Option  $i_number$  allows to print only i-th element of the selection. It is convinient in scripts. For atom selections it will also show full information about each atom, rather than only the ranges of atom numbers.

This string form is convenient used for several purposes:

- to store selections in tables and arrays.
- to transfer selections from object to object and from session to session (see also the Select function)

An example in which we generate text selection of the Crn leucine neighbors :

```
nice "lcrn"
nei = String( Res(Sphere( a_/leu a_/!leu , 4.)) )
show nei
a_lcrn.m/14:17,19:20
display xstick $nei
```

Another example with a loop over atom selection of carbon atoms:

```
read pdb "2ins"
for i=1, Nof( a_//c* )
    print String( a_//c* i )
endfor
```

# 2.21.123. Chemical formula

String([all|dot|smiles|sln]as\_)

- returns string with the following chemical information:

option	description	example
all	chemical formula	e.g. C2H6O
dot	chemical formula with dot-separated molecules	e.g. C2H6O.C3H8

smilessmiles stringslnsln notationSee also:Smiles, smiles.

e.g. [CH3][CH2][OH] e.g. CH3CH2OH

Example:

build string "se ala" # alanine show String(a\_//\* all ) # returns chemical formula: C3H5NO show String(a\_//\* sln ) # returns SLN notation: NHCH(CH3)C=O show String(a\_//\* smiles) # returns SMILES string: [NH][CH]([CH3])C=O

### 2.21.124. Sstructure

secondary structure function.

Sstructure (*rs*\_) – returns string of secondary structure characters ("H","E","\_", etc.) extracted from specified residues *rs*\_.

Sstructure ( { *rs\_* | *s\_seqStructure* } compress ) – returns the compressed string of secondary structure characters, one character per secondary structure segment, e.g. HHE means helix, helix, strand. Use the Replace function to change B to '\_' and G helices to H helices, or simply all non H,S residues to coil (e.g. Sstructure(Replace(ss,"[!EH]","\_"),compress) )

Example:

```
show Sstructure("HHHHHHHH____EEEEE",compress) # returns string "HE"
#
read object "crn"
show Sstructure( a_/A , compress) # returns string "EHHEB"
```

Sstructure ( { seq\_ | s\_sequenceString } ) - returns string of secondary structure characters ("H", "E","\_"). If this string has already been assigned to the sequence seq\_ with the set sstructure command or the make sequence ms\_ command, the function will return the existing secondary structure string. To get rid of it, use the delete sstructure command.

Alternatively, if the secondary structure is not already defined, the Sstructure function will predict the secondary structure of the *seq\_* sequence with the Frishman and Argos method.

If the specified sequence is not a part of any alignment of sequence group only a single sequence prediction will be effected (*vide infra*). Otherwise, a group or an alignment will be identified and a true multiple sequence prediction algorithm is applied. The multiple sequence prediction by this method reaches the record of 75% prediction accuracy on average for a standard selection of 560 protein chains under rigorous jack-knife conditions. The larger the sequence set the better the prediction. Prediction accuracy for a single sequence is about 68%. To collect a set perform the **fasta** search (Pearson and Lipman, 1988) with ktup=1 and generate a file with all the sequences in a fasta format.

#### Method used for derivation of single sequence propensities.

Seven secondary-structure related propensities are combined to produce the final prediction string. Three are based on long-range interactions involving potential hydrogen bonded residues in anti-parallel and

parallel beta sheet and alpha-helices. Other three propensities for helix, strand and coil, respectively, are predicted by the "nearest neighbor" approach (Zhang et al., 1992), in which short fragments with known secondary structure stored in the database (icmdssp.dat) and sufficient similarity to the target sequence contribute to the prediction. Finally, a statistically based turn propensity (also available separately via the Turn(*sequence*) function), is employed over the 4-residue window as described by Hutchinson and Thornton (1994). The function also returns four real arrays in the M\_out matrix [4, *seqLength*]. There arrays are:

- M\_out[1] : alpha-helix propensity [0., 1.]
- M\_out[2] : beta-sheet propensity [0., 1.]
- M\_out[3] : coil propensity [0., 1.]
- M\_out[4] : prediction reliability [0., 1.]

Note that these propensities are not directly related to the prediction. Usually the reliability level of 0.8 guaranties prediction accuracy of about 90%. Do not be surprised if the propensities are all zero for a fragment. It may just mean that the statistics is too scarce for a reliable estimate.

Examples:

### 2.21.125. Sum

function.

Sum (*iarray*) – returns the integer sum of iarray elements.

Sum ( { rarray | map }) - returns the real sum of elements.

Sum (*matrix*) – returns the rarray of sums in all the columns.

Sum (*sarray* [*s\_separator*]) – returns string of **concatenated** components of a sarray separated by the specified *s\_separator* or blank spaces by default. See also the opposite function: Split.

Examples:

```
show Sum({4 1 3})  # 8
show Sum(Mass(a_1//*))  # mass of the first molecule
show Sum({"bla" "blu" "bli"})  # "bla blu bli" string
show Sum({"bla" "blu" "bli"},"\t")  # separate words by TAB
show Sum({"bla" "blu" "bli"},"\n")  # create a multiple line string
```

### 2.21.126. Symgroup

function.

Symgroup ( { *s\_groupName* | *os\_object* | *m\_map* } ) - returns the integer number of one of 230 named space groups defined in ICM.

Symgroup ({ *i\_groupNumber* | *os\_object* | *m\_map* } string) - returns the string name of one of 230 space groups defined in ICM.

Symgroup (*i\_groupNumber*) – returns the rarray of transformation matrices (12 numbers each) describing symmetry operations of a given space group.

Examples:

iGroup = Symgroup("P212121") # find the group number=19
print "N\_mol. in the cell =", Nof(Symgroup(iGroup))/12

### 2.21.127. Table

generic function return a table.

```
Table(s_URL_encoded_String[crypt])
```

- returns the table of "name" and "value" pairs organized in two string arrays. The URL-encoding is a format in which the HTML browser sends the HTML-form input to the server either through standard input or an environmental variable. The URL-encoded string consists of a number of the "name=value pairs separated by ampersand ( & ). Additionally, all the spaces are replaced by plus signs and special characters are encoded as hexadecimals with the following format %NN. The Table function decodes the string and creates two string arrays united in a table.

Option crypt allows to interpret doubly encoded strings (e.g. ' is translated to '+' which then converted into a hexadecimal form). Frequently the problem can be eliminated by specifying the correct port. Example: you need to set a="b c" and d="<%>". Normal server will convert it to a=b+c> Double encoding leads to a=b%2bcE. To parse the last string, use the crypt option.

To see all the hidden symbols (special attention to '\r'), set l\_showSpecialChar =yes.

Examples:

```
read string  # read from stdin in to the ICM s_out string
a=Table(s_out)  # create table a with arrays a.name and a.value
show a  # show the table
for i=1,Nof(a)  # just a loop accessing the array elements
  print a.name[i] a.value[i]
endfor
See also: Getenv( ).
```

### 2.21.128. Converting alignment into a table

Table (*ali*\_[number]) – returns the table of relative amino acid positions for each of the sequence in alignment *ali*\_. Gaps are marked by zero. The first column of the table, .cons, contains sarray of consensus characters. All the other arrays are named according to the sequence names by default, or by the sequential number of a sequence in the alignment, if option number is specified. The table may be used to project numbers from one sequence to another. See also the Rarray(R\_, ali\_, seq\_) function. This table may look like this:

#>T pos		
#>-conss	seq1	-seq2
	0	1
	0	2
C	1	3
	2	0
~	3	4
С	4	5
	0	6
# for the fo	ollowing	alignment:
# Consensus	C ~C	
seql	CYQC-	_
seq2	LQC-NCI	P
	~ -	

Example:

```
read alignment "sh3"
t = Table(sh3 number) # arrays t.1 t.2 t.3
t = Table(sh3) # arrays t.cons t.Fyn t.Spec t.Eps8
#
cc = t.cons ~ "[A-Z]" # all the conserved positions
show cc # show aa numbers at all conserved positions
show t.Fyn>=10 t.Fyn<=20 # numbers of other sequences in this range</pre>
```

### 2.21.129. Extracting parameters of stack conformations

Table(stack[vs\_])

- return table of parameters for each conformation in a stack. If a variable selection argument is provided, the values of the specified variables are returned as well.

```
% icm
 buildpep "ala his trp"
 montecarlo
 show stack

      blow Statk

      iconf>
      1
      2
      3
      4
      5
      6
      7

      ener>
      -15.1
      -14.6
      -14.2
      -13.9
      -11.4
      -1.7

      rmsd>
      0.3
      39.2
      48.0
      44.1
      27.4
      56.6
      39.3

      paft
      1
      2
      2
      2
      1
      1
      1
      1

                1
                              0
                                            0
 naft>
                                                          1
                                                                       1
4
                                                                                      1
                                                                                                     0
                                   1
                                                1
 nvis>
                     4
                                                              4
                                                                                        4
                                                                                                          1
t= Table(stack)
show t
 #>T t
 #>-ener----rmsd-----naft-----nvis-----
      -15.126552 0.295555 1
                                                        4
      -14.639667 39.197378
                                            0
                                                                    1
```

```
-14.572973 47.996203 0
                              1
   -14.220515 44.058755 1
                              4
   -13.879041 27.435388 1
                              4
   -11.438268 56.636246 1
                              4
   -1.654792 39.265912 0
                              1
t1= Table(stack v //phi,psi) # show also five phi-psi angles
\# > T
#>-ener----rmsd--naft-nvis-----v1-----v2-----v3------v4-----v5-----
   -15.12
        0.29 1 4
                      -79.10 155.59 -75.30 146.99 -141.13
   -14.63 39.19 0 1 -157.22 163.56 -78.25 139.51 -137.30
  -14.57 47.99 0 1 -157.26 166.87 -85.08 92.55
                                               -84.74
```

See also: Iarray(stack) function

### 2.21.130. Tan

tangent trigonometric function. Arguments are assumed to be in degrees.

Tan ( {  $r_Angle | i_Angle \}$  ) – returns the real value of the tangent of its real or integer argument.

Tan (*rarray*) – returns rarray of the tangents of each component of the array.

#### Examples:

show Tan(45.)	# 1.
show Tan(45)	# the same
<pre>show Tan({-30., 0. 60.})</pre>	<pre># returns {-0.57735, 0., 1.732051}</pre>

#### 2.21.131. Tanh

hyperbolic tangent function.

```
Tanh ({ r_Angle | i_Angle \}) – returns the real value of the hyperbolic tangent of its real or integer argument.
```

Tanh (*rarray*) – returns rarray of the hyperbolic tangents of each component of the array.

Examples:

#### 2.21.132. Tensor

function the second moments for a multidimensional distribution.

Tensor  $(M_{-})$  – returns the square matrix of second moments of **K** points in **N** –dimensional space,  $M_{ki}$  (k=1,K,i=1,N). The matrix NxN is calculated as

 $\langle X_i \rangle \langle X_i \rangle - \langle X_i X_j \rangle$ , where  $\langle ... \rangle$  is averaging over a column k=1,K, and i,j=1,N.

- In one-dimensional case, *N*=1, when *M*\_ is just one column (*k*=1,*K*; *i*=1,1) the function returns a one by one matrix with the mean-square-deviation of the vector (which is equal to Rmsd(R\_)\*Rmsd(R\_)).
- N=2, x and y dimensions; In this case the function returns the 2 by 2 matrix: with  $\langle x \rangle^2 \langle x^2 \rangle$  and  $\langle y \rangle^2 \langle y^2 \rangle$  on the diagonal and  $\langle x \rangle \langle y \rangle \langle xy \rangle$  off-diagonal elements.
- In three-dimensional case the function returns three by three **tensor of inertia** (it was too tiring to type the formula in html). This matrix is useful for superposition of bodies or molecules on the basis of shape, since three principal coordinates can be easily derived from the tensor using the Eigen or Disgeo functions. This trick used in the \_dockScan script.

Example: buildpep "AAA" # a long molecules  $xyz = Xyz(a_//c^*)$  # a coordinate matrix of carbons # you can also do it with grobs:  $xyz = Xyz(g_myGrob)$  a=Tensor(xyz) # compute 3 by 3 matrix of the second moments b=Eigen(a) # returns 3 axis vectors ax1=b[?,1] # this is the longest half axis ax2=b[?,2] # this is the second half axis ax3=b[?,3] # this is the shortest half axis len1 = Length(ax1) # long axis length len2 = Length(ax2) # mid axis length len3 = Length(ax3) # short axis length r = Matrix(3,3) # to make the rotation matrix from b normalize the axes r[?,1] = ax1 / Length(ax1) r[?,2] = ax2 / Length(ax2) r[?,3] = Vector(r[?,1], r[?,2]) rotate a\_ Transpose(r) # rotates the principal axes to x,y,z # x the longest

This commands are assembled in the calcEllipsoid M\_xyz macro which returns ellipseRotMatrix, and three vectors: ellipseAxis1, ellipseAxis2 and ellipseAxis3

See also: Rot, rotate, transform

Example to orient the principal axes of the molecule along X,Y and Z (the longest axis along X, etc.).

### 2.21.133. Temperature

function returning the oligonucleotide duplex melting temperature.

```
Temperature ( { s_DNA_sequence | seq_DNA_sequence } [ r_DNA_concentration_nM [ r_Salt
concentration_mM ] ] )
```

- returns the real melting temperature of a DNA duplex at given concentration of oligonucleotides and salt. The temperature is calculated with the Rychlik, Spencer and Roads formula (Nucleic Acids Research,

v. **18**, pp. 6409–6412) based upon the dunucleotide parameters provided in Breslauer, Frank, Bloecker, and Markey, Proc. Natl. Acad. Sci. USA, v. 83, pp. 3746–3750. The following formula is used:

Tm=DH/(DS + R ln(C/4)) -273.15 + 16.6 log[K+]

where DH and DS are the enthalpy and entropy for helix formation, respectively, R is the molar gas constant and C is the total molar concentration of the annealing oligonucleotides when oligonucleotides are not self-complementary. The default concentrations are C=0.25 nM and [K+] = 50 mM. This formula can be used to select PCR primers and to select probes for chip design. Usually in primer design the temperatures do not differ from 60. by more than several degrees.

#### 2.21.134. Time

function returning time spent in ICM.

Time (*string*) – returns the string of time (e.g. 00:12:45) spent in ICM.

Time () – returns the real time in seconds spent in ICM.

Examples:

if (Time( ) > 3660.) print "Tired after " Time(string) " of work?"

### 2.21.135. Tolower

convert to the lowercase.

Tolower (*string*) – returns the string converted to the lowercase. The original string is not changed

Tolower (sarray) - returns the sarray converted to the lowercase. The original sarray is not changed.

Examples:

```
show Tolower("HUMILIATION")
```

read sarray "text.tx" #create sarray 'text' (file extension is ignored)
text1 = Tolower(text)

See also: Toupper().

### 2.21.136. Torsion

angle function.

Torsion (*as*\_) – returns the real torsion angle defined by the specified atom as\_ and the three previous atoms in the ICM-tree. For example, Torsion(a\_/5/c) is defined by { a\_/5/c , a\_/5/ca , a\_/5/n , a\_/4/c } atoms. You may type: print Torsion( and then click the atom of interest, or use GUI to calculate the angle.

Torsion (as\_atom1, as\_atom2, as\_atom3, *as\_atom4*) – returns the real torsion angle defined by four specified atoms.

Examples:

### 2.21.137. Toupper

convert to the uppercase.

Toupper (string) - returns the string converted to the uppercase. The original string is not changed

Toupper (sarray) - returns the sarray converted to the uppercase. The original sarray is not changed.

Examples:

```
show Toupper("promotion")
```

read sarray "text.tx"
text1 = Toupper(text)

See also: Tolower().

### 2.21.138. Tr123

translate one-character sequence to three-character notation.

Tr123 (sequence) - returns string like "ala glu pro".

Examples:

```
show Tr123(seq1)
```

See also: Tr321(). IcmSequence().

### 2.21.139. Tr321

translate three-character sequence to one-character notation.

 $Tr321(s_)$  – returns sequence from a string like this: "ala glu pro". This function is complementary to function Tr123(). Unrecognized triplets will be translated into 'X'.

Examples:

show Tr123("ala his hyp trp") # returns AHXT

#### 2.21.140. Trace

matrix function.

Trace (*matrix*) – returns the real trace (sum of diagonal elements) of a square matrix.

Examples:

show Trace(Matrix(3)) # Trace of the unity matrix [3,3] is 3.

### 2.21.141. Trans

translation function. 3D translation vector or DNA sequence translation.

```
Trans (R_12 transformationVector) – extracts the R_3 vector of translation from the transformation vector.
```

```
Trans ( seq_DnaOrRnaSequence )
```

- returns the translated DNA or RNA sequence ('-' for a Stop codon, 'X' for an ambiguous codon) using the standard genetic code. See also: Sequence( *seq\_*reverse ) for the reverse complement DNA/RNA sequence.

Example (6 reading frames):

```
w=Sequence("CGGATGCG>>AAATGATGCTGTGGCTCTTAAAAAAGCAGATATTGGAG")
show Trans(w), Trans(w[2:999]),Trans(w[3:999])
c=Sequence(w,reverse)
show Trans(c), Trans(c[2:999]),Trans(c[3:999])
```

Trans ( seq\_DnaOrRnaSequence { all | frame } [ i\_minLen] [ s\_startCodons] )

return a table of identified open reading frames in DNA sequence not shorter than  $i\_minLen$ . The function was designed for very large finished sequences from the genome projects. Currently the Standard Genetic code is used. Option  $s\_startCodons$  allows to provide a comma–separated list of starting codons; if omitted, the default is "ATG", another example would be "ATG, TTG" (for S.aureus).

Option frame indicates that both start and stop codons need to be found. If they is not found or the fragment is too short, the table will be empty.

Option all allows to translate ALL POTENTIAL peptides by assuming that the start and/or stop codons may be beyond the sequence fragment. In this case, initially all 6 frames are produces. Later, some of them can be filtered out by the *i\_minLen* threshold. The unfinished end codons will be marked by 'X'.

The table has the following structure:

- frame integer 1 2 3 for the direct chain, or -1, -2, -3 for the complementary chain, respectively
- left translation offset in the direct strand (even if translation occurred in the complementary chain)
- right translation offset in the direct strand.

- dir direction (+1 for the direct, –1 for the complementary)
- len fragment length
- seq sequence string

For example, if the fragment is in the complementary strand it may have the following parameters:

<pre>#&gt;-frame</pre>	left	right	dir	len	seq
-1	22	57	-1	12	XCVXVAAESVAS

In this case translation follows the reverse strand (frame=-1), starts in position 57 of the original direct sequence and proceeds to position 22.

Example:

```
dna=Sequence("TTAAGGGTAA TATAAAATAT AAAGTTCGAA CAATACCTCA CTAGTATCAC AACGCATATA")
T=Trans(dna frame 10)
sort T.left
show T
```

#### 2.21.142. Transpose

matrix function.

Transpose (matrix) - converts the argument matrix[n,m] into the transposed matrix [m,n]

Transpose (rarray) - converts real vector [n] into a one-column matrix [n,1]

Examples:

Transpose(a) # least squares fit Transpose({1. 2. 3.}) # [3,1] matrix

#### 2.21.143. Trim

function to trim array/matrix/string.

Trim (*I\_iarray i\_lower i\_upper*) – returns iarray clamped into the specified range. Values smaller than *i\_lower* are replaced with *i\_lower*, and values greater than *i\_upper* are replaced with *i\_upper*.

Trim (*R\_rarray r\_lower r\_upper*) – returns rarray clamped into the specified range.

Trim (*i\_i\_lower i\_upper*) – returns integer clamped into the specified range (e.g. Trim(6,1,3) returns 3).

Trim (r\_r\_lower r\_upper) - returns real clamped into the specified range.

Trim (*M\_matrix r\_lower r\_upper*) – returns matrix clamped into the specified range.

Trim (*s\_string* [all]) – returns string with removed trailing blanks and carriage returns. If option all is specified, both leading and trailing blank characters will be removed.

Trim (*s\_iarray s\_list\_of\_allowed\_characters*) – returns string with all characters except for the listed in the second argument are removed. Example:

```
Trim("as123d","abcds")
   asd
```

Trim (*s\_string i\_maxLength*) – returns string which is truncated if it is longer than the *i\_maxLength* argument.

Trim (*S\_sarray*) – returns sarray of strings with removed trailing blanks.

#### 2.21.144. Turn

beta-turn prediction function.

```
Turn ( { seq_{-} | rs_{-} } ) - returns rarray containing beta-turn prediction index. The index is derived from propensities for i,i+1,i+2,i+3 positions for each amino-acid. Pi = pi+pi1+pi2+pi3, then high Pi values are assigned to the next three residues. The propensities are taken from Hutchinson and Thornton (1994).
```

Examples:

```
s = Sequence("SITCPYPDGVCVTQEAAVIVGSQTRKVKNNLCL")
plot comment=String(s) number Turn(s) display # plot Turn prediction
```

See also the predictSeq macro.

#### 2.21.145. Type

generic function returning type.

```
Type ( icm_object_or_keyword ) - returns a string containing the object type (e.g. Type(4.32) and
Type(tzWeight) return string "real"). The function returns one of the following types:
"integer", "real", "string", "logical", "iarray", "rarray", "sarray",
"aselection", "vselection", "sequence", "alignment", "profile", "matrix", "map", "grob",
"command", "macro", "unknown".
```

```
Type (as_, 1) – returns a string containing the level of the selection ("atom", "residue", "molecule", "object").
```

```
Type (os_object, 2) – returns a string containing the os_object (or current by default) molecular object type. Defined types follow the EXPDTA (experimenal data) card of PDB file with some exceptions, see below:
```

"ICM"	ready for energy calculations. Those objects are either built in ICM or converted to the ICM-type.
"X-Ray"	determined by X-ray diffraction
"NMR "	determined by NMR
"Model"	theoretical model (watch out!)

"Electron"	determined by electron diffraction
"Fiber"	determined by fiber diffraction
"Fluorescence"	determined by fluorescence transfer
"Neutron"	determined by neutron diffraction
"Ca-trace"	upon reading a pdb, ICM determines if an object is just a Ca-trace.

"Simplified" special object type for protein folding games.

The non-ICM types can be converted to "ICM" with the convert command or convertObject macro.

Type ( {  $ms_| rs_|$ , 2) – returns the string type of the specified molecule or residue. Legal types are "Amino", "Hetatm", "Nucl", "Sugar", "Lipid", "empty". Residues of the "Amino" type can be selected with the 'A' character (e.g. a\_/A). See also a one-letter code for the type which is used in selections, ( e.g. a\_A, H). Examples:

Type (as\_ { atom | mmff } ) - returns an iarray containing the ICM or MMFF atom types. Example:

Type  $(as_1 as_2)$  – returns an integer containing the covalent bond type between the selected atoms.

Type (*seq\_*, 2) – returns the string type of the sequence. Two types are recognized: "protein" and "nucleotide". An example in which we rename and delete all DNA sequences from the session:

```
read pdb sequence "ldnk"
Type( ldnk_b, 2)
nucleotide
```

read pdb sequence "1dnk" Type( 1dnk\_b, 2) nucleotide

```
for i=1,Nof(sequence)
    if Type(sequence[i],2) == "nucleotide" rename sequence[i] "dna"+i
endfor
delete sequence "dna*"
```

#### 2.21.146. Unix

the output of a UNIX command.

Unix (*s\_unix\_command*) – returns the string output of the specified unix command. This output is also copied to the s\_out string. This function is quite similar to the unix command. However, the function, as opposed to the command, can be used in an expression.

Examples:

```
show Unix("which netscape") # equivalent to 'unix which netscape'
#
```

if ( Nof(Unix("ls"),"\n") <= 1 ) print "Directory is empty"</pre>

#### 2.21.147. Value

values of bond lengths, bond angles, phase and torsion angles.

Value (*vs\_var*) – returns rarray of selected parameters. The function considers variables only in the current object.

Examples:

```
ang=Value(a_/14:50/phi,PSI)  # array of phi-psi values
hbonds=Value(a_//bh*)  # array of lengths for all H-X bonds
```

(Note use of special torsion PSI in the first example.)

#### 2.21.148. Vector

vector product between two 3D-vectors.

#### **Vector product**

Vector (*R\_vector1 R\_vector2*) – returns rarray [1:3], which is the vector product with components

 $\{v1[2]^*v2[3] - v2[2]^*v1[3], v1[3]^*v2[1] - v2[3]^*v1[1], v1[1]^*v2[2] - v2[1]^*v1[2]\}$ 

#### Vector symmetry transformation

Vector ( $M_{matrix}$ ) – transforms an augmented affine 4x4 space transformation matrix into a transformation vector.

See also: Augment() function.

#### 2.21.149. Version

information about version of the current executable.

Version ([full]) – returns string containing the current ICM version. The second field in the string specifies the operating system: "UNIX" or "WIN". At the end there is a list of one-letter specifications of the licensed modules separated by spaces (e.g. " G B R ").

Option full adds a few fields:

- The compilation date of the executable, e.g. [Mar 18 2002 14:48]
- the path of the current executable
- the names of the licensed modules spelled explicitly

See also: show version.

Example:

show Version() # it returns a string if (Real(Version()) < 2.6) print "YOUR VERSION IS TOO OLD" if (Field(Version(),2) == "UNIX") unix rm tm.dat if (Field(Version(),2) == "WIN") unix del C:\tm\tm.dat if Version() == " D " print " Info> the Docking module license is ok"

#### 2.21.150. Volume

volumes of grobs, spheres, residues and cells.

Volume (*grob*) – returns real volume of a solid graphics object (would not work on dotted or chicken wire grobs). ICM uses the Gauss theorem for calculate the volume confined by a closed surface:

 $\mathbf{V} = 1/3 * \mathbf{Sum}(\mathbf{A} * \mathbf{n} * \mathbf{R})$ 

where A is a surface area of a triangle, **n** a normal vector, and **R** is is a vector from an arbitrary origin to any vertex of the triangle. It is important that the grob is closed, otherwise, strictly speaking, the volume is not defined. However, small surface defects will not affect substantially the calculation. ICM minimizes possible error by rational choice of the origin, which is mathematically unimportant for an ideal case. To define directions of the normals the program either takes the explicit normals (i.e. they may be present in an input Wavefront file) or uses the order of points in a triangle. ICM-generated grobs created by the make grob [ potential | matrix | map] command have the correct vertex order (the corkscrew rule), while the make grob skin command calculates explicit normals. The best way to make sure that everything is all right is to display grob solid and check the lighting. If a grob is lighted from the outside, the normals point outwords, and the grob volume will be positive. If a grob is lighted from the inside (as for cavities), the normals point inwords and the volume will be seriously disappointed, but he will issue a fair warning from the grave. The surface area is calculated free of charge and is stored in r\_\_out.

Example:

See also: the Area( grob )} function, the split command and How to display and characterize protein cavities section.

Volume ( $r_radius$ ) – returns real volume of a sphere, (4/3)Pi\*R<sup>3</sup>

Volume (*s\_aminoAcids*) – returns real total van der Waals volume of specified amino-acids.

Volume ( $R\_unitCellParameters$ ) – returns real volume of a cell with parameters {a,b,c} for a parallelepiped or {a, b, c, alpha, beta, gamma} in a general case.

Volume (g\_grob) – returns real volume confined by a grob.

Examples:

```
vol=Volume(1.)  # 4*Pi/3 volume of unit sphere
vol=Volume("APPGGASDDDEWQSSR")  # van der Waals volume of the sequence
vol=Volume({2.3,2.,5.,80.,90.,40.})  # volume of an oblique cell
```

#### 2.21.151. View

parameters of the graphics window and graphics view.

View ([window]) – returns rarray of 36 parameters of the graphics window and view. With the **window** option the function returns only *WindowWidth* and *WindowHeight*.

- first 16 numbers: rotation matrix and perspective.
  - ♦ V[1] V[5] V[9] screen X-axis (from left to right)
  - ◆ V[2] V[6] V[10] screen Y-axis (from bottom to top)
  - ◆ V[3] V[7] V[11] screen Z-axis (outwords into the screen)
- V[17:32] (next 16 numbers): the model view matrix, view point, scale and cliping planes
- V[33:34] XleftPos YupperPos
- V[35:36] WindowWidth WindowHeight. Size is given in pixels, Y is measured from the top down.

See also set view and set view.

Example (how to save the image with 3 times larger resolution):

```
nice "lcrn" # resize window
write image window=2*View(window) # 2-times larger image
```

View (R\_36\_FromView, R\_36\_ToView, r\_factor)

- returns rarray of the **interpolated view** between the from and to view at the intermediate point 0  $r_{factor1}$ . If  $r_{factor}$  is out of the [0,1] range, the operation becomes extrapolation and should be used with caution. The camera view is changed in such a manner that the physical space is not distorted (the principal rotation is determined and interpolated, as are translation and zoom)

Example:

```
nice "lcrn" # manually rotate and zoom
rl= View() # save the current view
# INTERACTIVELY CREATE ANOTHER VIEW
r2= View() # save the new view
for i=1,100 # INTERPOLATION
  set view View(r1,r2,i*0.01)
endfor
```

See also View(), set view and set view.

```
View({ "x"|"X"|"y"|"Z"|"Z"}) - returns rarray of 3 coordinates of the specified axis of the screen coordinate system.
```

Example:

```
build string "se ala"
display # rotate it now
show View("x")
gl=Grob("ARROW",3.*View("x"))
display gl
```

#### 2.21.152. Warning : the ICM warning message

indicates that the previous ICM-shell command has completed with warning. Warning() - returns logical yes if there was an warning in a previous command (not necessarily in the last one). After this call the internal warning flag is reinstalled to no. Warning(string) - returns string with the last warning message. In contrast to the logical Warning() function, here the internal warning code is **not** reinstalled to 0, so that you can use it in expressions like if Warning() print Warning(string).

Example:

```
read pdb "2ins" # has many warnings
if Warning() s_mess = Warning(string) # the LAST warning only
print s_mess
```

#### 2.21.153. Xyz : atom coordinates and surface points

function returning a matrix of x,y,z coordinates.

Xyz(as\_)

- returns matrix [number\_of\_selected\_atoms, 3] in which each row contains x,y,z of the selected atoms.

Examples:

```
coord=Xyz(a_//ca)# matrix of Ca-coordinatesshow coord[i]# 3-vector x,y,z of i-th atomshow Mean(Xyz(a_//ca))# show the centroid of Ca-atoms
```

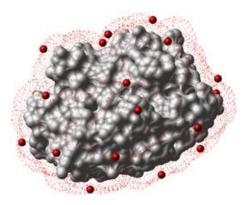
```
Xyz ( as_ r_interPointDistance surface )
```

- returns a subset of representative points at the accessible

surface which are spaced out at approximately *r\_interPointDistance* distance. This distance from the van der Waals surface (or skin) is controlled by the vwExpand parameter.

#### Example:

```
buildpep "ala his trp"
vwExpand = 3.
mxyz = Xyz( a_ 5. surface )
display skin white
dsXyz mxyz
color a_dots. red
```



# 2.22. Macros

Macros provide you with a great mechanism to create and develop your ICM environment and adjust it to your own needs (see also How do I customize my ICM environment.). Very often a repeated series of ICM commands is used for dealing with routine tasks. It is wise not to retype all these commands each time, but rather to combine them into a bunch for submission as a single command. Several examples follow.

#### 2.22.1. buildpep: Building peptides from a sequence

buildpep s\_seq

creates a new ICM object from an input sequence. This macro recognizes if you specified the sequence in one-letter upper-case letters or lower-case three-letter code and adds uncharged N- and C- termini. Use semicolon (;) to separate molecules. If you want to use different termini, or build a non-peptide molecule apply the build command directly or modify the macro.

For a multimolecular object you can also create separate objects and then move them together.

Examples:

buildpep "ala his trp glu" # one tetrapeptide: nter and cooh added buildpep "ala his ; trp glu" # two di-peptides buildpep "one ; one" # two oxygens buildpep "YTGSNVKVAV" # decapeptide buildpep "AQSVPYGVSQ;IKAPALHSQG" # two decapeptides

# 2.22.2. calcBindingEnergy: estimates electrostatic, hydrophobic and entropic binding terms

```
calcBindingEnergy ms_1 ms_2 s_terms ("el,sf,en")
```

evaluates energy of binding of two complexed molecules  $ms_1$  and  $ms_2 s_terms$  for the given set of energy terms  $s_terms$ . This macro uses the boundary element algorithm to solve the Poisson equation. The parameters for this macro have been derived in the Schapira, M., Totrov, M., and Abagyan, R. (1999) paper.

Example:

```
read object s_icmhome+"complex"
cool a_
calcBindingEnergy a_1 a_2 "el,sf,en"
```

#### 2.22.3. calcDihedral4atoms: calculate a torsion angle defined by four atoms

calcDihedral4atoms as\_1 as\_2 as\_3 as\_4

calculates an angle between the two planes specified by any four atoms, *as\_1 as\_2 as\_3 as\_4*. Usually these are four consequtive covalently bound atoms.

Example:

386

```
buildpep "ala his his"
display atom label
calcDihedral4atoms a_/3/nd1 a_/3/cg a_/3/cd2 a_/3/ne2
Angle= -0.06781 deg. (also saved in r_out) # it is almost flat
```

# 2.22.4. calcDihedralAngle: calculate an angle between two planes in a molecule

calcDihedralAngle as\_plane1 as\_plane2

calculates an angle between the two planes specified by two triplets of atoms, specified by the *as\_plane1* and *as\_plane2* selections

An example in which we measure an angle between planes of two histidines:

```
buildpep "ala his his" # we use another macro here
display atom labels
calcDihedralAngle a_/2/cg,nd1,cd2 a_/3/cg,nd1,cd2
Angle= 131.432612 deg. (in r_out).
```

# 2.22.5. calcEnsembleAver: Boltzmann average the energies of the stack conformations

calcEnsembleAver r\_temperature s\_parameter

a macro showing an example of how to calculate a Boltzmann–weighted average given a conformational stack of conformation representatives. The stack may be formed as a result of a Monte Carlo simulation or created manually. The *s\_parameter* string contains any expression returning the parameter to be averaged (e.g. "Value( $v_{2/2}$ phi)" or "Distance( $a_{2/2}$ ca  $a_{4/2}$ a)").

Example:

buildpep "ala his his" set vrestraint a\_/\* # impose rotamer probabilities mncallsMC = 5000 montecarlo # a stack is formed with energies calcEnsembleAver 300. "Value(v\_/2/phi)"

See also macro helicity.

#### 2.22.6. calcMaps: calculate five energy maps and write them to files

calcMaps s\_fileNameRoot ("rec") R\_box (Box (a\_5.)) r\_gridSize (0.5)

calculates five energy grid maps for the current object with the grid size r\_gridSize in the 3D box volume defined by the R\_box. The maps are saved to files with names *s\_fileNameRoot\_gc.map s\_fileNameRoot\_gh.map* etc. and are deleted upon return from the macro. Be careful with selecting a box. You may focus the box on the area of interest (e.g. Box( a\_/55,66 , 7.)). To use the maps read them in, rename to m\_gc m\_gh, etc. and set terms "gc,gh,ge,gb,gs". If you determined the box interactively you may just use the Box() function without arguments (it returns the parameters of the graphical box).

Example:

read object s\_icmhome+"crn"
calcMaps "crn" Box( a\_/15 4. ) 0.6
read map "crn\_ge"
rename m\_crn\_ge m\_ge
display m\_ge {1 2 3 0 4 5 6}
# the maps can be used in another session

# 2.22.7. calcPepHelicity: calculate average helicity of a peptide from movie frames

calcPepHelicity s\_movieName r\_temperature (300.)

a macro showing an example of how to calculate the helicity of a peptide structure given an ICM movie of the conformations accepted during a Monte Carlo run. A simulation using montecarlo movie option is a prerequisite for this macro. A good script prototype can be found in the *\$ICMHOME/\_folding file*. The movie option saves each accepted conformation to a movie file. The secondary structure of all transient conformations is assigned with the assign sstructure command.

Example:

```
% _folding # run the _folding script with the movie option.
% icm
```

read object "mypep" # the name of your peptide object calcPepHelicity "mypep" 600.

See also macro calcEnsembleAver

# 2.22.8. calcProtUnfoldingEnergy: rough estimate of solvation energy change upon unfolding

calcProtUnfoldingEnergy ms\_(a\_1) i\_mncalls(100)

calculates an octanol/water transfer solvation energy for the given # conformation as compared to an extended chain conformation.

# 2.22.9. calcRmsd: calculate three types of Rmsd between protein conformations

calcRmsd  $rs_1$  (a\_1.1/\*)  $rs_2$  (a\_2.1/\*)

calculates Ca-atom, backbone-atom, and heavy-atom RMSD for two input residue selections. The main effort in this macro is to take the internal symmetry of amino-acid sidechains into account.

For example, two phenylalanines related by the 180 degrees rotation of the xi2 angle are identical, but will have a non-zero  $\text{Rmsd}(a_1./\text{phe} a_2./\text{phe})$  because cd1 and ce1 of one selection lay on top of cd2 and ce2 atoms of the second selection, respectively. To calculate this Rmsd correctly, we need to find the rotation The following residues have internal symmetry (or pseudo-symmetry): leu, tyr, phe, asp, glu, arg, val.

#### 2.22.10. calcSeqContent

calcSeqContent s\_seqNamePattern ("\*")

analyzes amino acid composition of the input sequence or sequences. Specify quoted sequence name, pattern (e.g. "\*\_HUMAN") or "\*" for all sequences.

Example:

```
read sequence s_icmhome+"seqs"
calcSeqContent "*" # matches names of all sequences
..
Statistics for 3 sequence(s): Azur_Alcde Azur_Alcfa Azur_Alcsp
AA N % Expected
A 42 10.34 7.85
C 9 2.22 2.55
...
calcSeqContent "*de" # sequences ending with 'de'
Statistics for 1 sequence(s): Azur_Alcde
Res N % Expected
A 20 13.42 7.85
C 3 2.01 2.55
D 8 5.37 5.17
```

#### E 6 4.03 6.95

The columns are as follows:

- 1. One-letter amino-acid code
- 2. The total occurrence of the amino acid
- 3. Relative percentage occurrence in the given set of sequences
- 4. Expected mean occurrence of the amino acid in proteins

 $\label{eq:loss} convertObject auto \textit{ms}_(a_) \textit{l_delete\_water} (yes) \textit{l_optimize\_hydrogens} (no) \textit{l_replace\_the\_original} (no) \textit{l_display} (no)$ 

converts a non-ICM object into and ICM object and performs some additional refinements. The macro returns  $r\_residialRmsd$  value containing the Rmsd of the model atoms from the equivalent template atoms (the same value is returned by the convert command in  $r\_out$ ). If this residual is greater than 0.5, it usually means some problems with the conversion (e.g. unusual residues, missing parts, etc.).

clusterChem s\_inObjects ("\*.ob") s\_outObject ("clustered.ob")

performs clustering based on chemical similarity

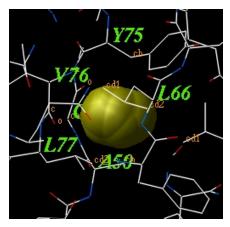
#### 2.22.11. icmCavityFinder: analyze and display cavities

icmCavityFinder as\_(a\_1) l\_interactive (no) r\_minVolume (3.)

calculates and displays cavities in a molecular structure. These cavities are sorted by size, and displayed. The  $l_interactive$  argument allows cavities to be displayed one by one interactively. To display the transparent outer shell edit the macro and activate this feature.

The *r\_minVolume* parameters defines the volume of the smallest retained cavity. Increase it if you want only large cavities.

For each cavity this macro calculates volume V (in square Angstoms), area A and an effective radius R (compare it with the radius of a water molecule of 1.4A).



The icmCavityFinder macro uses two powerful features of ICM-shell:

- a grob with analytical molecular surface (a.k.a. skin) of the selected atoms can be built using the make grob skin as\_ as\_ "g\_skin" command.
- this grob can be divided into the separate grobs with the outer shell and all the inner cavities with the split g\_skin command.

• icmCavityFinder also uses the Volume(g) and Area(g) functions to measure volume and area of the cavities, as well as the Sphere( g\_ r\_radius ) function to select atoms and residues around any grob.

Example:

```
read object s_icmhome+"1qoc"
delete a_w* # remove water molecules
icmCavityFinder a_1 yes 4.
   3 icm-objects deleted
 Info> finished surface search, n_of surface atoms = 744
 Surface .....
 Info> finished basic surface element calculations
 Info> Estimated vertex number = 335800, actual = 184896
 Info> packing vertices... sorted... done!
 Info> skin grob "g_skin" created (solid model: 32197 point
 Info> 3 grobs g_skin1 ... g_skin3 created
Shell 1: V=11039.291805 A=4525.876702
 Warning> Volume(g_skin2) may be improperly calculated: env
CAVITY 2: V=25.253718 A=44.282805 R~1.710848 ------
- Num Res. Type ---- SS Molecule ---- Object - sf - sfRati
     26 ile Amino IH m
                                                                       1qoc 0.0 0.00

      20
      11e
      Amino
      1
      In
      In

      53
      leu
      Amino
      L
      E
      m

      58
      val
      Amino
      V
      _
      m

      76
      val
      Amino
      V
      E
      m

      87
      val
      Amino
      V
      E
      m

      89
      ile
      Amino
      I
      E
      m

        lqoc
        0.0
        0.00

        lqoc
        0.0
        0.00
```

#### 2.22.12. dsCellBox: displays crystallographic unit cell

dsCellBox os\_

displays unit crystal cell box for the specified object *os*\_generated according to crystal symmetry parameters. This tiny macro extracts the cell from the object using the Cell function and makes a grob out of this array with the Grob function.

```
macro dsCellBox os_ (a_)
gCell = Grob ("cell" Cell(os_))
display gCell magenta
keep gCell
endmacro
```

See also: dsCell

#### 2.22.13. dsCell: cell and crystallographic neighbors

```
dsCell os_(a_) l_deleteRemoteNeighbors (yes) i_Ncells (4)
```

displays copies of the input molecular object generated according to crystal symmetry parameters.

#### 2.22.14. dsCharge: one of many ways to show charge residues

#### dsCharge

displays CPK representation of positively and negatively charged amino acid residues in red and blue colors, respectively. See also macro undsCharge

```
macro dsCharge
display a_*./asp,glu/o?* cpk red
display a_*./lys,arg/nz,n?* cpk blue
endmacro
```

#### 2.22.15. dsChem : chemical style display

dsChem *as*\_(a\_)

3D display of the input atom selection in chemical style and on white background.

If you want to 'flatten' the molecule you can perform a procedure from the following example:

```
buildpep "trp"  # you need an ICM object
tzMethod = "z_only"  # tether to the z-plane
set tether a_  # each atom is tethered to z=0
minimize "tz"  # keep the cov. geometry
```

#### 2.22.16. dsConsensus: 3D display of conserved residues

```
dsConsensus ali_seq_l_dsLabels (yes) s_graphType ("skin")
```

identifies and displays a consensus sequence pattern on the surface of a molecular structure.

#### 2.22.17. dsCustom: extended display and property-coloring

dsCustom as\_(a\_//\*) s\_dsMode("wire") s\_colorBy("atom") l\_color\_only(no)

Displays the specified representation ("wire", "cpk", "ball", "stick", "xstick", "surface", "ribbon") of a molecular selection and colors the selection according to the following series of features:

- atom type ( s\_coloringType="atom" ),
- residue type ("residue"),
- unique molecules ("molecule"),
- secondary structure type ("sstructure"),
- N-to-C-terminal chain course (NtoC""),
- B-factors ("bfactor"),
- electric charges ("charge"),
- solvent accessibility ("accessibility"),
- residue polarity ("polarity"),
- residue hydrophobicity ("hydrophobicity")

# 2.22.18. dsCustomFull macro for molecular display

dsCustomFull *as*\_(a\_//\*) *s\_display\_mode* ("wire") *s\_color\_by* ("atom") *l\_noWater* (yes) *l\_areaSelfMode* (no) *l\_color\_only* (no)

an extension of the previous dsCustom macro which, in addition, allows to color by an external rarray of 26 elements for each character. This user-defined array may contain any residue property information.

Flag  $l_areaSelfMode$  determines if the surface area is calculated for the selection in the context of all the atoms of the object (no) or only the selection itself, as if no other atoms existed (the *self* mode)

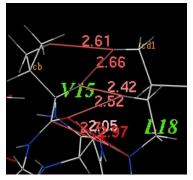
#### 2.22.19. dsDistance: display distances between two selections

dsDistance *as\_1 as\_2 r\_lowerLimit* (0.) *r\_upperLimit* (3.)

displays distances in a specified range between atoms of two input atom selections. This macro saves atom names and distances in T\_dist table. This table can later be resorted and analyzed.

Example:

read object s\_icmhome + "crn"
dsDistance a\_/15 a\_/18 0. 10.
show T\_dist



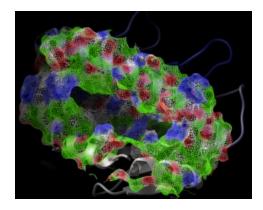
# 2.22.20. dsPropertySkin: display molecular surfaces colored by properties essential for binding

dsPropertySkin *as\_sel* (a\_) *l\_wire* (yes)

displays essential properties of molecular surfaces which are essential for binding small ligands, peptides or other proteins.

The first argument is a selection of atoms involved in the surface calculation. The second argument allows you to display the surface as:

- skin(l\_wire=no), or
- wire (l\_wire=yes)



The color code:

- white neutral surface
- green hydrophobic surface
- red hydrogen bonding acceptor potential
- blue hydrogen bonding donor potential

Example shown:

```
read pdb "1a9e"
delete a_w*
convert # convert to ICM for map calculations
    # select receptor atoms 9. away from the peptide with Sphere
cool a_ # display ribbon
dsPropertySkin Sphere( a_3 a_1 9. ) yes
    # adjust clipping planes for better effect
write image png
```

Interactive surface display under GUI The same can be performed interactively on ICM objects with the popup-menu:

- 1. display your ICM object
- 2. switch selection level to **residue** (R)
- 3. select region with selection box or lasso
- 4. click on the right mouse button over one of the selected residues
- 5. selected Display and then Property Skin
- 6. it creates grob g\_recSkin which can then be undisplayed and further manipulated

# 2.22.21. dsEnergyStrain: analyzing energy strain in proteins

dsEnergyStrain rs\_(a\_/A) l\_colorByStrain (yes) R\_limits ({-2., 5.})

calculates relative energy of each residue for residue selection  $rs_{}$ ; and colors the selected residues by strain (if logical *l\_colorByStrain* is "yes"). The *R\_limits* argument determines the range represented by the color gradient (i.e. residues strained beyond 5. will still be shown in red).

This macro uses statistics obtained in the Maiorov, Abagyan, 1998 paper.

Example:

```
read object s_icmhome + "crn"
dsEnergyStrain a_/A yes {-2., 5.}
#>-S1-----eStrain----
" T1 " -0.438031
" T2 " 1.500011
" C3 " 0.171743
....
```

### 2.22.22. dsEnergyStrain1

```
dsEnergyStrain1 rs_l_colorByStrain (yes) l_sstruct
```

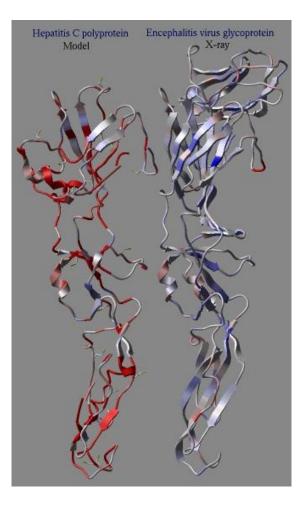
another version of above. If 1\_sstruct in on, the type of the secondary structure is taken into account for energy strain calculations.

#### 2.22.23. icmPmfProfile

```
macro icmPmfProfile os_(a_)
l_accessibilityCorrection (yes) l_display (no)
```

calculates a statistical energy of mean-force for each residue of a provided object. This energy is calculated with the "mf" parameters defined in the icm.pmf file. The residue energies are then normalized to the expected mean and standard deviation of the same residue in real high resolution structures. The mean energy value can be calculated as a function of its solvent accessibility if *l\_accessibilityCorrection* is set to yes.

The calculated table contains residue energies and accessibilites. These values can be used to color residues of the molecule according to those values. In an example shown here we build a model (using the build model command of HCV protein on the basis of another viral coat protein. Then the profile was calculated for the model and the original structure. The calculation clearly shows the problematic regions of the model (the red parts) while the source structure looks quite reasonable.



#### 2.22.24. dsPrositePdb

dsPrositePdb ms\_(a\_\*) r\_prositeScoreThreshold (0.7) l\_reDisplay (no) l\_dsResLabels (yes)

Finds all PROSITE pattern-related fragments in the current object and displays/colors the found fragments and residue labels.

#### 2.22.25. dsRebel: surface electrostatic potential

dsRebel ms\_l\_assignSimpleCharges

generates the skin representation of the molecular surface colored according to the electrostatic potential calculated by the REBEL method (hydrogen atoms are ignored). The coloring is controlled by the maxColorPotential parameter. This macro uses a simplified charge scheme (a\_/lys/nz : 1.0, a\_/arg/nh\*: 0.5, a\_/asp,glu/oe\*,od\*: -0.5) and uses only the heavy atoms for the calculations for the sake of speed. A full atom version of this macro is dsRebel.

#### 2.22.26. dsSeqPdbOutput : visualize the sequence similarity search results

dsSeqPdbOutput s\_searchSeqPdbOutput ("tm.ou")

Goes through a list of PDB hits resulting in find database command and displays alignment(s) of the input sequence(s) with the found PDB structures and SWISSPROT annotations.

#### 2.22.27. dsSkinLabel

dsSkinLabel rs\_s\_color("magenta")

For all residues specified by the input residue selector, rs\_, displays residue labels shifted toward the user to make the labels visible when skin representation is used.

#### 2.22.28. dsSkinPocket and dsSkinPocketIcm

dsSkinPocket *ms\_ligand* (a\_2) *ms\_receptor* (a\_) *r\_radius* (7.): \*dsSkinPocketIcm ms\_ligand (a\_2) ms\_receptor (a\_1) r\_radius (7.)

display the receptor pocket around the selected ligand *ms\_ligand*. Only the largest contiguous pocket surrounding the ligand is retained for clarity. The dsSkinPocketIcm macro also colors the molecular surface by hydrogen bonding potential and hydrophobicity. Best used with the ligand shown in cpk, if the ligand is small.

These macros can also be used to show the protein-protein interface.

Example:

read object s\_icmhome+"complex"
cool a\_
dsSkinPocket a\_1 a\_2 7. # shows the surface of a\_1

#### 2.22.29. dsStackConf

dsStackConf as\_

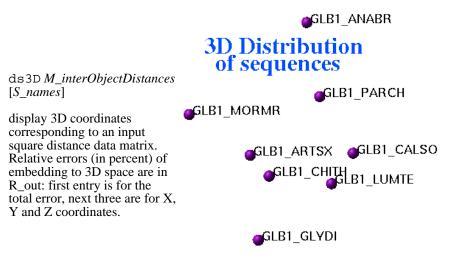
displays superimposed set of conformations from a conformational stack for given selection as\_.

#### 2.22.30. dsVarLabels

dsVarLabels

displays color labels for different types of torsion variables.

#### 2.22.31. ds3D



Representation of inter-sequence evolutionary distances in three-dimensional space

#### 2.22.32. dsWorm

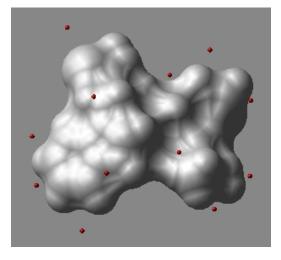
dsWorm ms\_

displays "worm" (or "tube") representation of selected molecule(s). Residue colors are smoothly changed from blue (at N-terminus) to red (at C-terminus).

#### 2.22.33. dsXyz : display

dsXyz M\_xyz\_coordinates

displays points from the  $N_atoms \ge 3$  matrix of  $M_xyz\_coordinates$  in 3D space as blue balls. The origin of the Nx3 matrix is not important. The macro creates an object called a\_dots. In this object each dot is a one-atom residue called 'dot'. The atom type is arbitrarily assigned to oxygen, and the atom names are 'o'.



One can further manipulate this object, e.g. color  $a_{12:15/o}$  green.

An example in which we generate sparse surface points at vwExpand distance around a molecule and display them.

```
buildpep "ala his trp"
mxyz = Xyz( a_ 5. surface )
display skin white
dsXyz mxyz
color a_dots. red
```

#### 2.22.34. findFuncMin

findFuncMin *s\_Function\_of\_x* ("Sin(x)x -1.") *r\_xMin*(-1.) *r\_xMax*(2.) *r\_eps*(0.00001)

minimizes one-dimensional functions provided as a string with the function expression. The macro uses successive subdivision method, and assumes that the function derivative is smooth and has only one solution in the interval

Example:

```
findFuncMin "Sin(x)*x-1." , -1. 2. 0.00001
    -1.000000 < x < 0.500000
    -0.250000 < x < 0.500000
    -0.062500 < x < 0.125000
    ...
    -0.000004 < x < 0.000008
    -0.000004 < x < 0.000002</pre>
```

# 2.22.35. findFuncZero

findFuncZero *s\_Function\_of\_x* ("Cos(x)x") *r\_xMin*(1.) *r\_xMax*(100.) *r\_eps*(0.00001)

finds a root of the provided function of one variable with specified brackets with iterations. E.g.

```
findFuncZero "x*x*x-3.*x*x" 1. 33. 0.00001
=> x=17.000000 F=4046.000000
=> x=9.000000 F=486.000000
=> x=5.000000 F=50.000000
=> x=3.000000 F=0.000000
```

### 2.22.36. nice

```
nice s_PdbFileName ("lcrn") l_wormStyle (no)
```

reads and displays a PDB structure in ribbon representation; colors each molecule of the structure by colors smoothly changing from blue (at N-terminus) to red (at C-terminus).

Example:

```
nice "365d" # new DNA drug prototype
nice "334d" # lexitropsin, derivative of netropsin
```

# 2.22.37. cool

cool auto rs\_sel (a\_)

similar to the macro nice above, but refers to a residue selection.

#### 2.22.38. homodel

```
homodel ali_l_quick (yes)
```

homology modeling macro. The first sequence in the input alignment should contain the sequence of a PDB template to which the modeling will be performed. If flag l\_quick is on, only an approximate geometrical model building is performed. You can also use the build model command directly.

#### 2.22.39. makeIndexChemDb

```
makeIndexChemDb s_dbFile ("/data/acd/acd.mol") s_dbIndex ("/inx/acd") s_dbType
("mol") S_dbFields ({"ID"})
```

Creates and saves an index to a small compound database existing in standard mol or mol2 formats (specified by the s\_dbType parameter). s\_dbIndex defines full-path root name of several index-related files. String array S\_dbFields specifies fields of the input database which are indexed by the macro.

An example in which we index the cdi.sdf file and generate the cdi.inx file in a different directory:

```
% icm
makeIndexChemDb "/data/chem/chemdiv/cdi.sdf" "/data/icm/inx/cdi" "mol" {"ID"}
```

# 2.22.40. makeIndexSwiss

```
makeIndexSwiss s_swiss ("/data/swissprot/fseq.dat") s_indexName
(s_inxDir+"/SWISS.inx")
```

Creates and saves an index to the SWISSPROT sequence database (datafile s\_swiss). s\_indexName defines the root name of several index-related files with respect to ICM user directory, s\_userDir.

#### 2.22.41. makePdbFromStereo: restore 3D coordinates from a stereo picture

transforms two stereo sets of two-dimensional coordinates in arbitrary scale into 3D coordinates. See also: How to reconstruct a structure from a published stereo picture .

#### 2.22.42. mkUniqPdbSequences

mkUniqPdbSequences i\_percentDifference (5) s\_seq\_dir s\_pdb\_dir (s\_pdbDir+"") l\_replace (no)

Creates a collection of PDB sequences with specified degree of mutual dissimilarity, i\_percentDifference. Replace old dataset if l\_replace is on.

# 2.22.43. plot2DSeq

plot2DSeq ali\_

generates a 2D representation of "distances" between each pair of sequences from the input alignment.

#### 2.22.44. plotSeqDotMatrix

plotSeqDotMatrix seq\_1 seq\_2 s\_seqName1 ("Sequence1") s\_seqName2 ("Sequence2") i\_mi
(5) i\_mx (20)

generates an EPS file in which local sequence similarities between two sequences are shown in the form of a two-dimensional dot-matrix plot. Significance of local sequence similarities is shown by logarithm of the probability values and is calculated in multiple windows from i\_mi to i\_mx. The log-probability values are color-coded as follows: light blue: 0.7, red 1.0.

# 2.22.45. plotSeqDotMatrix2

plotSeqDotMatrix2 seq\_1 seq\_2 s\_seqName1 ("Sequence1") s\_seqName2 ("Sequence2") i\_mi
(5) i\_mx (20)

generates an EPS file in which local sequence similarities between two sequences are shown in the form of a two-dimensional dot-matrix plot. Significance of local sequence similarities is shown by (1. - Probability(..)) values and is calculated in multiple windows from i\_mi to i\_mx. The (1 - P) values are color-coded as follows: light blue: 0.7, red 0.99.

# 2.22.46. plotBestEnergy

plotBestEnergy s\_McOutputFile ("f1,f2") r\_energyWindow (50.)

plots profile of energy improvement during an ICM Monte Carlo simulation. Data are taken from the MC output log file or files, s\_McOutputFile. You can specify a single output file (e.g. "f1.results"), or several files, e.g. "f1.ou, f2.ou", or drop the default ".ou" extension, e.g. "f1, f2, f2".

This macro gives you an idea about the convergence between several runs.

#### 2.22.47. plotOldEnergy

```
plotOldEnergy s_McOutputFile ("f1.ou") r_energyWindow (50.)
```

plots profile of energy changes during an ICM Monte Carlo simulation. Data are taken from the MC output log file, s\_McOutputFile.

#### 2.22.48. plotFlexibility

```
plotFlexibility seq_i_windowSize (7)
```

calculates and plots flexibility profile for input sequence *seq\_* and smooths the profile with *i\_windowSize* residue window.

#### 2.22.49. plotCluster

```
plotCluster M_distances S_names ({""}) s_plotArgs ("CIRCLE display {\"Title\"
\"X\" \"Y\"}")
```

plot distribution of clusters. Arguments:

• a square matrix of distances between n objects. For arrays it may be calculated with the Distance () function (e.g. Distance(Xyz(a\_//ca))). For angular RMSD the distance can be calculated from a matrix **v** of values of torsion angles for many conformations:

• sarray of names for each of n points . Possibilities:

- the empty sarray: {""}. No name tags will be attached to each point
- ◆ Sarray(Count(n)) generates names like this: {"1" "2" "3" ... }
- ♦ user-defined: e.g. Name(a\_\*.) if each point correspond to an object

♦ manual: e.g. {"A" "compound X" "c"}

• See also: arguments for the plot command.

# 2.22.50. plotMatrix

plotMatrix M\_data s\_longXstring S\_titles ({"Title", "X", "Y"}) s\_fileName ("tm.eps")
i\_numPerLine (10) i\_orientation (1)

generates combined X-Y plot of several Ys (2nd, 3rd, etc. rows of the input matrix M\_data) versus the one X-coordinate, assumed to be the first row of the matrix. i\_numberPerLine parameter defines the size of the plotted block size if the number of data points is greater then i\_numberPerLine. i\_orientation equal to 1 defines portrait orientation of the output plot, landscape otherwise.

### 2.22.51. plotRama

plotRama rs\_l\_show\_residue\_label (no) l\_shaded\_boundaries (yes)

generates a *phi–psi* Ramachandran plot of an *rs*\_residue selection. If logical *l\_show\_residue\_label* is on, the macro marks the residue labels. If *l\_shaded\_boundaries* is on, the allowed (more exactly, core) regions are shown as shaded areas; otherwise the contours of the core regions are drawn.

# 2.22.52. plotRose

```
plotRose i_prime (13) r_radius (1.)
```

just a nice example of a simple macro generating "rose" plot.

# 2.22.53. plotSeqProperty

```
plotSeqProperty R_property s_seqString S_3titles {"Y property", "Position", "Y"} s_fileName
("tm.eps") i_numPerLine (30) s_orientation ("portrait")
```

a generic macro to plot local sequence properties. Modify it for your convenience. Here is an example in which we plot residue b-factors along with the crambin sequence. s\_seqString could be the sequence (e.g. String(lcrn\_m)) or secondary structure, (e.g. Sstructure(lcrn\_m)) or any other string of the same length as the sequence.

```
read pdb "lcrn"
make sequence
b = Bfactor( a_/* )
plotSeqProperty b String(lcrn_m ) {"" "" ""} "tm.eps" 20 "portrait"
```

# 2.22.54. predictSeq

```
predictSeq s_seq s_fileName l_predictSecStr
```

calculates and plots hydrophobicity and flexibility profiles and secondary structure diagrams for the given sequence *s\_seq* (this is a string with the sequence name) and saves the results in *s\_fileName* PostScript file.

# 2.22.55. prepSwiss

prepSwiss s\_IDpattern ("VPR\_\*") l\_exclude (yes) s\_file ("tm")

extracts all sequences from the SWISSPROT database which exclude (l\_exclude= yes) or include (l\_exclude= no) the specified sequence pattern, s\_IDpattern and creates a set of database files with the rootname s\_file intended to use in the command find database.

# 2.22.56. printFast

printFast s\_ofPrinterName ("graphic")

writes current content of the graphic window to a PostScript file and calls unix 'lp' command to send the resulting PostScript file to the specified printer, s\_ofPrinterName.

# 2.22.57. printMatrix

printMatrix s\_format(" %4.1f") M\_matrix (def)

prints matrix *M\_matrix* according to the input format *s\_format*.

#### 2.22.58. printPostScript

```
printPostScript s_ofPrinterName ("grants")
```

converts the current content of the graphics window to a PostScript file and directs it to the *s\_ofPrinterName* printer.

#### 2.22.59. printTorsions

printTorsions rs\_

outputs all torsion angles of the input residue selection.

# 2.22.60. refineModel: globally optimize side-chains and anneal the backbone

refineModel i\_numberOfAnnealingIters (5) l\_sideChainSampling (no)

This macro can be used to improve any ICM model. The model can come from the build model command or the convert command or regul macro, etc. It performs

- side-chain sampling using montecarlo fast
- interative annealing with tethers you have provided
- second side-chain sampling to resolve the new problems resulting from the second step

To perform only the side-chain refinement, set the *i\_numberOfAnnealingIters* argument to 0.

# 2.22.61. regul

regul rs\_(a\_/A) s\_regObjName ("regobj") s\_ngroup ("nh3+") s\_cgroup ("coo-") l\_newIcmSeq
(yes) l\_displayRegul (yes) l\_freeMin (no)

creates a regularized ICM-model of an input residue selection (rs\_) under the name s\_regObjName. If l\_newIcmSeq is set to yes, the macro will create sequence from that of the input residue selection, optionally modified by the N- and C-terminal groups (s\_ngroup and s\_cgroup, empty " " strings are allowed); otherwise the macro will use an ICM-sequence file, **s\_regObjName**.se The protocol course may be displayed if l\_displayRegul set to yes. The resulting ICM model will be written to file s\_regObjName.ob. If l\_freeMin set to \*yes\*, the resulting model will be additionally minimized, now without tethers, and be written to file s\_regObjNamef.ob. The summary of the macro's work will be saved to file s\_regObjName.log.

#### 2.22.62. rdBlastOutput

rdBlastOutput S\_giArray

reads a set of sequences defined in a BLAST's output file, S\_giArray from the NCBI database.

#### 2.22.63. rdSeqTab

rdSeqTab s\_dbase("NCBI")

reads a set of sequences listed in the ICM-table SR, an output of find database command, from the database defined by  $s\_dbase$ .

#### 2.22.64. readPdbList

readPdbList S\_list\_of\_pdb\_codes

reads a series of PDB files specified in the input string array and creates sequences for all loaded structures.

#### 2.22.65. remarkObj

remarkObj

allows editing an annotation (comment) of the current object. Existing comment (if any) is read in an editor and after modification assigned to the object.

#### 2.22.66. searchPatternDb

searchPatternDb s\_pattern ("?CCC?") s\_dbase ("SWISS")

searches for the pattern in the sequences of the specified indexed database  $s\_dbase$ .

# 2.22.67. searchPatternPdb

```
searchPatternPdb s_pattern
```

searches for the specified pattern in pdb sequences taken from the foldbank.db file.

Example (first hydrophobic residue, then from 115 to 128 of any residues, non-proline and alanine at the C-terminus):

```
searchPatternPdb "^[LIVAFM]?\{115,128\}[!P]A$"
```

#### 2.22.68. searchObjSegment

```
searchObjSegment ms_i_MinNofMatchingResidues (20) r_RMSD (5.)
```

for given molecule *ms*\_finds all examples of similar 3D motifs not shorter than *i\_MinNofMatchingResidues* residues with the accuracy *r\_RMSD* A in the ICM protein fold database.

#### 2.22.69. searchSeqDb

searchSeqDb s\_projName ("sw1") S\_seqNames ({ ""}) r\_probability (0.00001) l\_appendProj (no)
s\_dbase ("SWISS")

search the database s\_dbase using query sequence(s) specified in S\_seqNames. Found hits and their specs are collected in the output table file s\_projName.tab. If logical flag l\_appendProj is on data will be appended to the existing table. Similarity of hits to the query sequence(s) is controlled by parameter r\_probability (see Probability()).

#### 2.22.70. searchSeqPdb

searchSeqPdb s\_projName ("pdb1") r\_probability (0.01) l\_appendProj (no)

sequence search of all currently loaded sequences in the sequences of the proteins from the fold bank collection. Found hits and their specs are collected in the output table file s\_projName. If logical flag l\_appendProj is on data will be appended to the existing table. Similarity of hits to the query sequence(s) is controlled by parameter r\_probability (see Probability()).

#### 2.22.71. searchSeqPdb

searchSeqPdb s\_projName ("pdb1") r\_probability (0.01) l\_appendProj (no)

sequence search of all currently loaded sequences through all proteins from the collection s\_pdbDir+"/derived\_data/pdb\_seqres.txt.Z", a subset of PDB sequences with given degree of mutual dissimilarity. Found hits and their specs are collected in the output table file s\_projName. If logical flag l\_appendProj is on data will be appended to the existing table. Similarity of hits to the query sequence(s) is controlled by parameter r\_probability (see Probability()).

# 2.22.72. searchSeqSwiss

```
searchSeqSwiss seq_
```

Searches for homologues of the query sequence seq\_ in the SWISSPROT database.

#### 2.22.73. setResLabel

```
setResLabel
```

moves displayed atom labels to the atoms specific to each residue type.

### 2.22.74. sortSeq

sortSeq

sort sequences by their length and suggest outliers.

### 2.22.75. undsCharge

undsCharge

color display of the charged residues.

#### 2.22.76. makeSimpleModel

makeSimpleModel seq\_ali\_os\_

This macro rapidly builds a model by homology using simplified residues described in the residue library. Input data are the sequence of the model,  $seq_a$  and alignment ali\_ of the model's sequence with the template object  $os_a$ .

# 2.22.77. makeSimpleDockObj

```
makeSimpleDockObj[os_object][s_newObjName]
```

This macro builds an ICM object from simplified residues described in the residue library. The goal is to convert an all-atom molecular object into an object in simplified representation for fast docking calculations.

# 2.22.78. searchSeqProsite

```
searchSeqProsite seq_
```

compares input sequence against all sequence patterns collected in the PROSITE database.

Examples:

See also:

find pattern, find database pattern=s\_pattrn, find prosite.

# 2.23. Files

#### 2.23.1. \_macro. A collection of ICM macros.

This file contains a set of ICM macros. You can use them, modify them, or browse them to develop your own macros. \_macro is downloaded by the call \_macro command.

#### 2.23.2. \_startup. ICM startup file

This ICM script contains a set of commands issued automatically upon invoking ICM. The file will be searched for in the directory defined by the UNIX environmental variable ICMHOME. This location may be different from the \$ICMHOME directory. It allows users to share the ICM executable but have their individual \_startup files. **Important**: edit this file to customize your environment. A template to modify follows.

```
s pdbDir
              = "/data/pdb/" # set it to the place where PDB lives
pdbDirStyle = "pdblabc.ent" # style currently distributed by PDB
s_helpEngine = "icm"
                             # reasonable default, HTML-help is
                             # an alternative
                             # you may have your own PROSITE updated file
s_prositeDat = Getenv("ICMHOME")+"/prosite.dat"
                             # xpsview may be more standard
s_psViewer = "/usr/opt/bin/gs -q"
                             # better be accessible only for you
s tempDir = "/usr/tmp/"
#
                             # they will be read from $ICMHOME
read libraries
call _aliases
                             # by default it will be taken
                             # from the directory defined by
                             # environmental variable $ICMHOME
call _macro
                             # by default it will be taken
                             # from the directory defined by
                             # environmental variable $ICMHOME
```

print "...ICM startup file executed..."

#### 2.23.3. \_startCheck script

This script checks the presence of and access to the directories and files used by ICM and specified by some ICM-shell string variables. This script is recommended during customization of the

ICM.

### 2.23.4. foldbank.db

Bank of assigned secondary structures (foldbank.db) This text file may be created by \_mkSegmentLib script and contains secondary structures for a nonredundant set of protein chains. Description of fields:

- NA chain name ('m' usually stands for main or 'NO chain identifier')
- RZ resolution. NMR entries get 9.99 (they may be actually worse than that).
- ER all-atom RMSD-error upon PDB->ICM conversion. Beware of entries with ER > 0.5!!
- SE amino acid sequence as extracted from the structure (not SEQRES)
- SX authors' secondary structure assignment, all \_\_\_\_\_ if not provided (as in 1knt.m).
- SS automatically assigned by ICM secondary structure using modified Kabsh and Sander algorithm.

The commented field contains a serial number.

Example two entries:

NA RZ ER	355 4tpi.i 2.20 0.027 RPDFCLEP		ARIIRYFYNA EEEEEEEE EEEEEEE_	EEEEEEE	E	AKRNNFI B	_нннннннн	HH
•••	•							
##	364							
NA	1knt.m							
RZ	1.60							
ER	0.015							
SE	TDICKLPK	DEGTCRD	FILKWYYDPN	TKSCARFWYG	GCGGN	JENKFG	SQKECEKVC	A
SX								_
SS	_GGGG	B	_EEEEEEE	EEEEEEE	B	B	_нннннннн	_
•••	•							
	•							

# 2.23.5. Bank of protein folds (foldbank.seg)

This text file contains descriptions of segment (or vector) representations of protein three-dimensional structures.

Example:

sis.m scorpion insectotoxin i5a \_E\_H\_E\_E\_1 1 2 3 6 8 4 4 1 5 3 -1 2.50 1.11 -0.95 3.95 2.97 -3.10 5.41 -3.20 -10.98 13.48 -0.74 -12.05 11.31 4.10 -3.08 13.56 0.33 -0.05 5.08 -8.21 -5.66 4.15 -7.57 -8.58 8.77 -1.09 2.21 6.15 0.21 7.04

Each molecule is represented by a single line containing the following fields:

- sis.m : molecular selection (note 'm' is used as a chain identifier if the pdb-file has no chain information).
- scorpion insectotoxin i5a : long name (up to the 30th position)
- \_E\_H\_E\_E\_ : secondary structure of ( '\_' coil, 'E' extended, 'H' helix )
- 1 1 2 3 6 8 4 4 1 5 3 -1 : a segment list; the first integer indicates the format for the segment list, the last -1 is a terminator. There are two formats, indicated by numbers 1 and 0.
- 1 : concise : ResNumberOffset 1st\_SegmentLength 2nd\_Segment\_Length 3rd\_Segment\_Length ...
- 0 : full format (e.g. 0 74b 4 78b 8 86b 4 90b 5 –1 ) : *1st\_Res\_Number/char 1st\_Segment\_Length* 2nd\_Res\_Number/char 2nd\_Segment\_Length ...

\* 2.50 1.11 –0.95 ... : x,y,z for all reference points. The full format allows more complex residue numbering which may frequently be found in the pdb–entries (i.e. 4 5 6 8 9 9a 9b 9c 10 12 ..). The concise format is used for the regularly numbered molecules.

#### 2.23.6. Atom codes (icm.cod)

This text file contains description of (1) *atom types* and references to (2) MMFF types, (3) van der Waals types, (type 0 or 9 to ignore) (4) hydrogen bonding types, (type 1 means no H–bonds) and (5) hydration types (0 to ignore). The real numbers are atomic mass (6) and surface (7). The character (8) is used to define atom color. Free–format. Two example lines:

#	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	* Comment
#>	cd	mmff	vw	hb	hd	wt	sf	na	comment
cod	63	6	18	6	6	16.000	40.77	0	* o in r-c-oh (thr,ser)
cod	71	32	19	6	8	16.000	36.79	0	* o- in carboxylate ion
cod	92	12	61	1	1	35.453	133.8	Cl	* (MMFF)
cod	223	38	13	4	3	14.007	61.16	n	* pyridin nitrogen (MMFF)

# 2.23.7. Bond angle bending and improper torsion deformation parameters (icm.bbt)

This text file contains a factor (kcal/mole) and an equilibrium angle in degrees for the bond angle bending deformation energy for different types of angles.

```
# Type Factor OptAngle(a0). E=Factor*(a-a0)<sup>2</sup>
#
bbt 1 160.7000 115.0000 ca-c#-n
bbt 2 128.2000 120.5000 ca-c#=0
...
```

#### 2.23.8. Bond stretching parameters (icm.bst)

This text file contains a factor (kcal/mole) and an equilibrium bond length in Angstroms for the bond stretching energy for different types of bonds.

#	E=Fact				
#	Type	Factor	BondLengt	h(b0).	
#>	ity	eybs	eqbl	bt	comment
#					
bst	15	00.0	1.4530	1 cn	n-ca
bst	2 11	50.0	1.3250	1 cn	c#-n
bst	34	60.0	1.5300	1 cc	ca-c#

. . .

# 2.23.9. Conformational stack (\*.cnf)

This binary file contains descriptions of several conformations of the same molecule. You can not edit this file. The stack is automatically generated and saved in the course of Monte Carlo, or systematic search procedures. Alternatively the stack may be created directly by the store conf command. To read/write a stack use: read stack [s\_StackName] write stack [s\_StackName]

#### 2.23.10. Distance restraint types ( icm.cnt or \*.cnt )

The file describes legal types of drestraints to impose attraction or repulsion between atom pairs (e.g. NOE distance restraints derived from NMR data). This penalty term is called "cn". The system icm.cnt can be edited, however, user files (e.g. mydist.cnt) of the same format can be created and loaded with the read drestraint type command. The file contains:

#	type	weight	lower	upper	sharpness
#					4 special types for S-S bonds
ssSS1		10.0	2.04	2.04	10.0 # Sharp well for S-S dist.
ssSS2		2.0	2.04	2.04	1.0 # Wide well for S-S dist.
ssSC		5.0	3.052	3.052	10.0 # S -Cb distance
ssCC		3.0	3.855	3.855	10.0 # Cb-Cb distance
global	1	1.0	0.0	3.0	# a global drestraint
global	2	1.0	2.0	4.0	# a global drestraint
local	12	1.0	2.5	2.8	1.0 # a local drestraint

Both local distance restraints and global ones force two atoms to stay between the upper and lower boundaries, however, the local restraints diminish at large distances (similar to van der Waals interactions), whereas the global restraints grow bi–quadratically as deviation from the target distance range increases. You can have and read several \*.cnt files. If the type numbers overlap the previous types are redefined.

See also related commands: read drestraint type, show drestraint type, set drestraint type, make drestraint type.

#### 2.23.11. Distance restraints (\*.cn)

Contains list of atom pairs for which interatomic distances should be restrained according to specified types defined in a separate icm.cnt file. The .cn files are created by the user. Supplied icm.cn file is just an example.

#	ml1	rel	at1	ml2 re2	at2	cn_type
cn	crn	1 val	hg22	* 1 val	ca	1
cn	*	1 val	hg23	* 1 val	cg1	1
cn	*	2 gln	ca	* 1 val	hg11	1
cn	*	2 gln	ca	* 1 val	ha	1

Molecule name and residue number (e.g. 14, 25A, etc.) are normally used to find an atom. An asterisk instead of the molecule name means that only the residue number should be matched. See also: read drestraint, show drestraint, set drestraint, and make drestraint (un)display drestraint

# 2.23.12. Graphics objects (\*.gro)

This text file contains streams of POINT coordinates (i.e. a triple of floats in one line, preceded by an integer reference number), LINE descriptors (i.e. pair of integer numbers of recently described POINTS in one line) and/or TRIPLES (or TRIANGLES, i.e. triples of integer numbers of POINTS). The order of streams is arbitrary, provided that referenced POINTS are already described. Either LINES or TRIPLES can be omitted. Graphics objects can be read, written, displayed, or made from a 3D map.

Check content of other .gro files in you icm directory. ICM also understands the Wavefront obj-format (files \*.obj). You can create your own dot, wire or solid graphics objects either manually or automatically.

#### 2.23.13. ICM HTML help file ( icm.htm )

contains this manual

#### 2.23.14. Hydrogen bonding types ( icm.hbt )

A and B parameters for the  $A/r^{12} - B/r^{10}$  potential between HB donors and acceptors. See Nemethy et al. for reference.

Example lines:

#	i	j	В	A	E	r0	
#							
hbt	2	4	8244.0	32897.0	0.550	2.190	* n-hn
hbt	3	4	8244.0	32897.0	0.550	2.190	* o-hn

#### 2.23.15. Hydration parameters ( icm.hdt )

Parameters to calculate solvation energy based on atomic solvent–accessible surfaces (see solvation term). The file contains several sets (e.g. Eisenberg and McLachlan (1986), Wesson and Eisenberg (1992)) although only one of them is not commented out.

Example lines:

rwater 1.4000 # water radius used to roll around the molecule
#
 1 2 3 4 5
#
hdt 4 -0.0500 1.7000 0.0016 n+

1. reference type number

2. solvation energy density from vacuum-water transfer experiments for a given hydration type

- 3. solvation energy density from octanol-water transfer experiments for a given hydration type
- 4. radius used to calculate accessible surface
- 5. comment

#### 2.23.16. Configuration file (icm.cfg)

This file contains limits and memory requirements for ICM. It will be searched in the **current** directory ( . / ) first and, if not found, in the directory defined by the UNIX environmental variable \$ICMHOME or \$HOME/.icm/config/ directory ( \$USERPROFILE/.icm/config/ for Windows), if present.

You may edit the file and change the limits.

```
# ICM configuration file. Free format
# Mn stands for "Maximal Number of"
# Mx stands for "Maximal Size of"
BufferSpace 2097152 # ICM will not let you decrease BufferSpace less than 131072
MnResidueTypes 200
MnSequences
              20000
MnAlignments
             1500
MnProfiles
                 40
                200
MnGrobs
MnMaps
                 40
MnMacros
                400
XTermFont *-fixed-medium-*-*-*-24-* # to set font in the terminal window
          xwsh
                                     # default for SGI
Xterm
                                     # default for Linux and other UN*Xes
# Xterm
          xterm
```

#### 2.23.17. Colors ( icm.clr )

file contains default color and font settings. The default icm.clr file resides in the \$ICMHOME directory. The LIBRARY.clr variable defines the default path and name of the icm.clr file.

Keep your own color and graphics controls file in ~/.icm directory. Example of ~/.icm/user\_startup.icm file ( \$USERPROFILE/user\_startup.icm under Windows ):

Modify the file if needed. The following lines are recognized (free format):

```
# CONFIGURABLE GRAPHICS.mode translation table
#
     Use keywords Left Mid Right, Shift Ctrl Alt Dbl, At
#
     TopNN LeftNN RightNN BottomNN, where NN is a percentage of the zone
#
    Modes 0,3,4,5,14,15 require a hit in 'At' = (atom | grob)
#
     otherwise control falls through to next best appropriate action
#
     Some modes have submode switches listed in parentheses ()
#
    Users are encouraged to modify bindings to their needs
# ---mode--combination------ # equivalent GRAPHICS.mode preference
mode0Right-At#popup (in GUI ofmode1(Shift)-Left#Rotationmode2(Shift)-Mid#Translationmode3(Ctrl)-Shift-Right-At#Label atomsmode4(Ctrl)-Dbl-Right-At#Label residues
                                          # popup (in GUI only)
mode 5 (Shift)-Ctrl-Left-At
                                         # Change torsion angles
```

```
6 (Shift)-Bottom5-Left
mode
                                    # Rotation of the view
mode 7
         (Shift)-Top5-Left
                                      # Z-axis rotation
mode 8 Left5-Mid
                                      # Zoom
mode 9 Alt-Mid
                                      # Move rear clipping plane
mode 10 Ctrl-Mid
                                     # Move front clipping plane
mode 11 Ctrl-Alt-Mid
                                      # Slab
mode 12 (Shift)-Right
                                      # Rectangular selection
mode 13 (Shift)-Ctrl-Left
                                      # Lasso selection
mode 14 (Shift)-Ctrl-Alt-Right-At # Connect to molecule
mode 15 Shift-Ctrl-Dbl-Right-At
                                      # Set alignment cursor
mode 16 Ctrl-Mid-At
                                      # Drag atoms
mode 17 Right5-Mid
                                      # Z-translate
#----- Colors ------
# -----color------RRGGBB--A real if not 1
. . . .
color lightgreen
                       # 80ff80
color rita
                        # ff1b00
                                    0.3
color darkseagreen # 8fbc8f
#----- Atom/Grob/Font Colors ------
atom c grey # c is the first character of chemical element.
background
             black
              color
                        font
                                   size bold italic underline
#
atomFont
             rose
                         times
                                    12 0
                                             0
                                                   0
varFont
             yellow
                         symbol
                                    12 0
                                              0
                                                   0

      helvetica 18
      0

      helvetica 18
      0

      times
      24
      0

      symbol
      28
      0

      courier
      12
      0

residueFont green
                                                   0
grobFont green
stringFont green
                                                  0
                                                  0
auxiliaryFont green
                                                  0
fixedFont green
                                                   0
#
alphaRibbon magenta
betaRibbon green
coilRibbon yellow
#- 0:127 rainbow colors (address them by number: color 15.5) ------
# -----i-color----
rainbow 0 # 0000ff
rainbow
           63 # ffffff
rainbow 127 # ff0000
```

## 2.23.18. Electron density map (\*.map)

This binary file contains a complete description of the electron density map, compatible with the format devised by Phil Evans. Maps are stored as a 3–dimensional array preceded by a header which contains all the necessary information about the map. See "The CCP4 Suite" manual for details.

## 2.23.19. MC simulation movie (\*.mov)

This binary file contains a description of a set of geometric parameters (free variables, usually torsions and overall rotation/translation variables), participating in MC simulation, followed by a stream of their values for each conformation accepted during the simulation, together with the energy of each accepted conformation. Movies can be created or appended during MC simulation runs, and then played in any direction with optional smoothing, superimposition (to the initial conformation) and/or centering. These files tend to be large, watch them carefully and do not create them without a need. See also: display

movie.

## 2.23.20. ICM-object (\*.ob)

a binary noneditable file describing one or several molecules forming an ICM–object. In addition to information available in a PDB–file it contains a description of atomic charges, tree–like connectivity, detailed atom codes, information about which internal coordinates are constrained, references to energy parameters, secondary structure, etc. The object can be read and written.

## 2.23.21. Residue library ( icm.res or \*.res )

The main residue library, describing all "residues" and molecules which can constitute a legal ICM-object. You can create your own entry either manually or using the write library command and add the entry to the icm.res file. You can also keep it in a separate file and append the file to the LIBRARY.res sarray (i.e. LIBRARY.res=LIBRARY.res//"usr" followed by the read library command). A example of an entry for a *pro* residue:

nare																			
rem																-			
rem				_ator	n	—,	_d1	.hec	dral_angl	es,	k	ono	d_angles_	—,	_bo	nd_	_lengti	າຮຸ	
rem		/				\	/			\	/				/			\	
rem	at		cd	lwat	qu	gu	na	fe	vuva	еy	na	fe	vuva	еy	na	fe	vuva	ey	qfm
#	Field																		
#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
#																			
atre	1	n	216	0	-0.285	0	psi	+	180.000	2	an		118.000	1	bn		1.340	31	
atre	2	са	113	1	0.050	0	omgr	+ (	180.000	24	aca		121.000	1	bca		1.465	1	
atre	3	ha	1	2	0.040	0	fha		116.800	0	aha		110.200	1	bha		1.090	0	
atre	4	cb	112	2	-0.025	0	fcb		-120.850	0	acb		103.700	37	bcb		1.530	4	
atre	5	hb1	1	4	0.015	0	fhb1		120.200	0	ahb1	L.	111.600	1	bhb1		1.090	0	
atre	6	hb2	1	4	0.015	0	fhb2	2.	-120.200	0	ahb2	2.	111.600	1	bhb2		1.090	0	
atre		cq	112	4	-0.050		xi1		27.400		acg		103.700		bcq		1.502	4	
atre		hq1	1	7	0.025	0	fhq1						111.200		bhg1		1.090	0	
atre		hg2	1	7	0.025	0			-120.450		ahg2		111.200		bhg2		1.090	Ō	
atre		cd	114	7	0.100		xi2		-35.600	21			105.300		bcd	•	1.501	4	
atre		hd1	5	10	0.010		fhd1	•	-119.730	0					bhd1	•	1.090	Ō	
atre		hd2	5	10	0.010		xi3		-90.760	-	ahd2		111.600		bhd2		1.090	Õ	
atre		C	121	2	0.455		phir	, .	-68.800				112.300		bc		1.520	3	
atre	14		81	13			fo	· ·	180.000		ao		120.500		bo		1.230	5	
#	F 20	0	01	10	0.505	-	10	·	100.000	0	uo	•	120.000	-	20	•	1.200	5	
lwat	13																		
±wac #		21																	
# exbo	1	10																	
EVDO	Ŧ	10																	

Eentropy is the entropic contribution to the free energy for a fully accessible residue divided by the solvent accessible surface of this residue (in gly gly X gly gly environment) and multiplied by a factor of 1000.

Fields:

- 1. relative atom number
- 2. atom name
- 3. main atom code (see icm.cod file). It in turn refers to other codes such as hydrogen bonding code, van der Waals code and hydration code.

- 4. previous atom in a connectivity graph of the directed ICM-tree.
- 5. electric charge
- 6. groups of close charges (they should not be separated due to cutoffs distance in interaction lists)
- 7. torsion name
- 8. fixation status (+ free variable, . fixed)
- 9. torsion angle (degrees)
- 10. torsion energy type (see icm.tot file).
- 11. bond angle name
- 12. bond angle fixation status
- 13. bond angle value (degrees).
- 14. bond angle deformation energy type (see icm.bbt file).
- 15. bond length name
- 16. bond fixation status
- 17. bond length (Angstroms)
- 18. bond stretching energy type (see icm.bst file).
- 19. (qfm) formal charge (may be +1, -1/3, -1/2 etc.), if any
- 20. (lwat) exit atom of the residue
- 21. (exbo) additional (non-tree) covalent bonds (atom1 atom2). Normal ICM-tree bonds form regular directed graph without cycles, therefore all the remaining bonds should be declared separately.

See also: LIBRARY.res.

## 2.23.22. Object Variables (\*.var)

Text file containing either a subset or a complete set of internal coordinates (variables). Usually created by the write vs\_var command. It may also be typed manually. In contrast to an object file (\*.ob) which is a complete description of an object, icm.var may contain any selection of variables. These variables can be read and automatically assigned to a molecule according to molecule name, residue number and variable name.

Examples:

rem Va	fx	Atom	Resi	due	Mol (	Obj	VaType	Sym	m Value
va psi	+	n	3	gln	mol	dl	2	1	180.00
va phi	+	С	3	gln	mol	dl	1	1	-60.00
va psi			4	met	mol	dl	2	1	120.00
va phi	+	С	4	met	mol	dl	1	1	180.00

## 2.23.23. Multidimensional variable restraint types (icm.rst or \*.rst)

define generic attraction zones in internal coordinate space (usually torsion space) in terms of residue name pattern (\* for any residue type), relative residue number, and variable names. After the types are loaded, you may use them to assign specific vrestraints using the set vrestraint command. Three example restraint types (the third one marked with 'rse' will be used as a penalty term, the first two will be used for BPMC steps):

------ Two-variable vrestraint rs aa -3.300 0.700 0.542 va ala\* 1 phi -63.200 22.500 va \* 2 psi -38.540 25.500 ------ One-variable vrestraint

rs vt		-3	.511	0.700	)	0.670
va val*	1	xi1	174.6	590	29.22	25
rse fmr		-3	.267	0.700	)	0.525
rse fmr va phe*						

Explanation of fields:

- **aa** rs–name (not longer than 4 characters). Usually the first character is a 1–char residue name, and the second is the zone character (a alpha, b–beta, g–gamma,d–delta,t–trans,p–plus 60, m–minus 60,n–null)
- -3.300 well depth ( should be negative )
- 0.700 flat fraction of the well
- **0.542** occupancy (probability) of the well with respect to other wells which could be assigned to the same set of variables.
- ala\* residue name pattern
- 1 relative residue number
- phi variable name
- -63.2 center of the well for 1 phi
- 22.5 size of the well (well is -63.2 + 22.5)
- \* 2 psi residue name pattern (\* means any), relative residue number (psi formally belongs to C atom of the next residue, that is why the relative number is 2) and variable name

*WARNING:* remember that the outer borders of the two–dimensional restraints are ellipses, rather than rectangles (the same for the multidimensional ones). To create a sloped surface for all variables involved in the restraint, the well sizes for these variables should be greater than 180.\*sqrt(number of angles in the rs) (255.0 for 2 angles).

#### 2.23.24. Multidimensional variable restraints (\*.rs)

This file has similar format to the icm.rst file. The differences are:

- rse and rs fields indicate whether the zone will be used for energy or probability, respectively.
- .rs file contains specific residue numbers rather than the relative ones.
- instead of the residue name pattern the file should contain molecule name or "\*".

Example:

	-3.300 0.7	Two-variable vrestraint
va 1crn 1	phi -63.200	22.500
va 1crn 2	2 psi -38.540	25.500
		One-variable vrestraint
rse vt	-3.511 0.7	700 0.670
va * 3	3 xil 174.690	29.225
rs fmr	-3.267 0.7	700 0.525
va 1crn 4	kxi1 -66.780	29.700
va 1crn 4	xi2 98.680	75.100

## 2.23.25. A sample \*.col file

The file shows an example of a multicolumn file which can be read with the read column command. Arrays *r*, *e s* and *ent* will be created.

```
# Entropies of several amino acids
#>-r--e-e----ent---
arg 2.13 184.920211 11.5181
asn 0.81 99.516352 8.1393
asp 0.61 89.629773 6.8057
cys 1.14 85.947233 13.263
gln 2.02 129.68481 15.576
glu 1.65 119.957333 13.75
his 0.99 121.124577 8.173
ile 0.75 132.717054 5.651
```

#### 2.23.26. A sample \*.tab file

The file shows an example of a .tab file which can be read with the read table. It is similar to the previous file but additionally command. Table t consisting of header string t.titl and arrays *t.r, t.e t.s* and *t.ent* will be created.

```
#>s t.titl
Entropies of several amino acids
#>T t
#>-r--e---s---ent---
arg 2.13 184.920211 11.5181
asn 0.81 99.516352 8.1393
asp 0.61 89.629773 6.8057
```

#### 2.23.27. Torsion parameters (icm.tot)

The file contains torsion parameters according to Momany et al., 1975. Parameters for type 21 for pro taken from Venkatachalam et al., (1974), Macromolecules, 7, 212, parameters for types 22–23 for cooh taken from Karplus et al., J.Comp.Chem., (1983),4,187–217, DNA parameters are from Veal and Wilson, 1991. Format is free.

# # #		maxEner	sign	fold		-	nmetry- Pseudo			ar	
tot	0	0.00	0	0	1	1	1	_	Ο.	#	fixed dihedrals
tot	2	0.25	1	1	1	1	1	S	90.	#	psi
tot	44	0.50	1	1	1	1	1	S	90.	#	psi beta preference
tot	3	10.00	-1	2	1	1	1	S	Ο.	#	omg
tot	4	1.35	1	3	1	1	1	Η	Ο.	#	xi CH2-CH2
tot	8	0.90	1	3	3	3	3	М	Ο.	#	nh3 term.group of lys,lysn
tot	14	0.00	0	2	1	1	2	Н	Ο.	#	xi2 of his (+-90)

Torsion energy is calculated as: maxEner\*(1 + sign\*Cos(fold \* torsion\_angle)) Symmetry is a rotational symmetry in different situations: **Exact** is the exact symmetry (implies presence of all atoms, including hydrogens), **Heavy** implies presence of only the heavy atoms (no hydrogens) but uniqueness of different atom types. **Pseudo** implies that all heavy atoms are equivalent, and hydrogens are ignored. The last character is a short reference name which can be used in vs\_var. For example: v\_//M specifies all the

torsions rotating terminal hydrogen atoms with symmetry higher than 1,  $v_{//H}$  side-chain torsions rotating heavy atoms, etc.

### 2.23.28. Van der Waals parameters ( icm.vwt )

The file contains ECEPP/2 parameters for peptides (Momany et al., 1975, Nemethy et al., 1983), parameters for DNA atoms: Veal and Wilson, 1991 and other parameters (unpublished).

Example lines:

#	type	pzat	n_el	energy	Deq	Rvw	Rvwel	electroRadii
#								
vwt	1	0.42	0.85	0.0370	2.92	1.200	1.200	* h aliphatic
vwt	2	0.42	0.85	0.0610	2.68	1.200	0.808	* h amide,amine
vwt	7	1.51	5.20	0.1400	3.74	1.700	1.700	* c carbonyl
vwt	39	0.00	0.00	0.55	5.911	2.631	2.631	* pseudo atom

Each line contains:

- 1. type reference number (see icm.cod file)
- 2. atomic polarizability  $*10^{24}$  (cm cubed);
- 3. effective number of electrons
- 4. -e(kk), kcal/mol depth of energy minimum at the optimal interatomic distance
- 5. r(kk), a equilibrium (optimal) distance between two atoms of the same type
- 6. van der Waals radius (used in graphics, electrostatics, etc.)
- 7. electrostatic radii, used to calculate geometrical surface boundary in boundary element method and MIMEL calculations

Normally van der Waals parameters *A* and *B* are calculated from polarizability and effective number of electrons (fields 2 and 3). However, if these two fields contain zeros, parameters *A* and *B* are calculated directly from the energy depth and equilibrium distance (e.g. for type 39).

## 2.23.29. Protein databank file ( or \*.ent )

Protein Data Bank formatted files consist of x,y,z coordinates, occupancies, and B-factors.

Examples:

ATOM	1	Ν	THR	1	17.047	14.099	3.625	1.00	13.79
ATOM	2	CA	THR	1	16.967	12.784	4.338	1.00	10.80
ATOM	3	С	THR	1	15.685	12.755	5.133	1.00	9.19

This file does not provide a complete and unambiguous description of a molecular object. Therefore an object resulting from the read pdb command has a special type and needs conversion in order to become a full-scale ICM-object for which energy calculations are possible.

See also: convert and minimize tether commands.

## 2.23.30. Sequence ( \*.seq \*.pir \*.gcg \*.msf \* )

Acceptable formats of sequence files:

• FASTA and ICM format (\*.seq the simplest and the most natural):

```
> Name1 comment1 comment2
AGFDSTREMNH-FQW
> Name2
RTPIYQWSCCVANMKL
```

• PIR format: (\*.pir)

```
>P1;Azur_Pses4
Length: 80
AECSVDIQGN DQMQFSTNAI TVDKACKTFT VNLSHPGSLP KNVMGHNWVL TTAADMQGVV
TDGMAAGLDK NYVKDGDTRV*
//
>P2;Azur_Pses3
Length: 50
AECSVDIQGN DQMQFSTNAI TVDKACKTFT VNLSHPGSLP KNVMGHNWVL*
```

• GCG format (difficult to generate and impossible to edit because of the CheckSum):

```
Azur_Alcfa Length: 69 Check: 4484 ..

1 ACDVSIEGND AMQFNTKSIV VDKTCKEFTI NLKHTGKLPK AAMGHNVVVS

51 DGMKAGLNND YVKAGDERV
```

• *MSF format* – obsolete multiple sequence format for alignments. Noneditable, contains CheckSums.

## 2.23.31. ICM-sequence file (\*.se)

contains molecular names and sequences. A simple example with two peptides:

```
ml a
se gly ala ser pro tyr his
se phe trp tyr
ml b
se ala ala ser asn
```

A more advanced example with numbering, N- and C-termini and D-amino acids:

```
ml subl
se 0 nter 1 gly 2 ala 2A Dglu 4 asp cooh
ml water
se 18 hoh
```

ml field followed by molecule name signals that a new molecule is started. se field indicates sequence lines (free format). Residue names should correspond to entries in the icm.res residue library. Residue numbers (if any) may be arbitrary, negative and may contain additional characters (e.g. 15A, 15B, etc.). Terminal modifiers (*nter, nh3+, cooh, coo-, conh,* etc.) may be explicitly specified.

## 2.23.32. ICM-alignment file

ICM-format for sequence alignments. The consensus string contains the following symbols:

symbol	description
space	gap in at least one of the sequences
character	this amino acid is conserved in all the sequences
+	positively charged amino acids (R,K)
_	negatively charged amino acids ( D,E )
^	small amino acids: ( A,S,G,S )
%	aromatic residues (F,Y,W)
#	hydrophobic amino acid (F,I,L,M,P,V,W)
~	polar amino acid ( C,D,E,G,H,N,Q,S,T,Y )
dot	the rest (no consensus, no gap)
The file l	ooks like this:

```
# comments
# Consensus:
Azur_Alcde
MLAKATLAIVLSAASLPVLAAQCEATIESNDAMQYNLKEMVVDKSCKQFTVHLKHVGKKAKVAMG
Azur_Alcfa
Azur_Alcsp
# Consensus: FCSFPGH#^#MKG.#
Azur_Alcde
FCSFPGHWAMMKGTLKLSN
Azur_Alcda
FCSFPGHWSIMKGTIELGS
Azur_Alcsp
FCSFPGHWAMKGVL----
```

## 2.23.33. ICM all-file: a file with multiple icm objects.

a file containing several ICM-shell objects divided by the following separators:

```
#> type1 ICM-shell-object-name1
.... obj1.....
.... obj1.....
#> type2 ICM-shell-object-name2
.... obj2.....
.... obj2.....
etc.
```

Legal separators:

- #>i integer\_name
- #>r real\_name
- #>s string\_name
- #>llogical\_name
- #>p preference\_name
- #>Īiarray\_name
- #>R rarray\_name

- #>S sarray\_name
- #>M matrix\_name
- #>seq sequence\_name
- #>prf profile\_name
- #>ali alignment\_name
- #>m map\_name
- #>g grob\_name
- #>T table\_name # the column layout
- #>coltable\_name

# the column layout

• #>db table\_name

# the database layout

- #> brk # a protein-data-bank file content
- #> var # internal variables (torsions, angles, bonds) for the current ICM-object

A sample file a.all containing an integer, real, logical, real array, a pdb-file and a table. You can read all this from a file or simply mark the lines and paste them into your ICM-session after the command: read all unix cat followed by Ctrl-D.

```
#>i numberOffset
  0
#>r lineWidth
  1.00
#>l logo
  yes
#>R boxx
 0. 0. 1. 1.
#>brk
           1 n
                                      2.602 -12.770 -6.750 1.00 20.00
ATOM
                   leu m
                             1
          2 ca leu m
3 cb leu m
4 cg leu m
5 cd1 leu m
                                      2.423 -11.442
                                                        -7.311
ATOM
                             1
                                                                 1.00 20.00
ATOM
                             1
                                      0.947 -11.187
                                                        -7.625
                                                                 1.00 20.00
                                                       -9.138 1.00 20.00
-9.649 1.00 20.00
-9.822 1.00 20.00
ATOM
                             1
                                      0.758 -11.068
ATOM
                             1
                                      1.487 -9.824
           6 cd2 leu m
                             1
                                      1.335 - 12.309
ATOM
#>s tt.h
this is a header string of table tt. The arrays follow.
#>i tt.i
15
#>T tt
#> a b c d
1 2. bla 13
3 5. bli
          13
```

## 2.23.34. Residue comparison table (icm.cmp or \*.cmp)

A triangular matrix with relative residue exchange frequences (see actual file). The amino acid character line serves as a ruler. Use your favorite comparison matrix.

## 2.23.35. Protein profiles (\*.prf)

A profile table contains residue preferences for each residue type in each sequence position. The preferences may be derived from a multiple sequence alignment or from three–dimensional structure.

Examples:

Cons A С E F etc. Z Gap Len .. В D С 35 -32 143 -42 -52 -12 etc. -62 100 100 Ρ 55 17 17 17 -71 etc. 26 6 100

## 2.23.36. Integer array (\*.iar)

File looking like this (free-format):

```
# everything which is not a number will be skipped
1 2 4
9
numbers may be in a row, or column or be in an arbitrary order.
-14 9
```

## 2.23.37. String array ( \*.sar )

Actually any text file. Each line will be a separate element of a string array

## 2.23.38. Matrix (\*.mat)

File looking like this:

 $\begin{array}{cccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{array}$ 

or like that

# my matrix
0. 1. 1. 2.2
1. 0. blu 1. 2. # text will be skipped
# 1. 1. 0. 3. this line is commented out

In the latter case the result of read matrix command is a matrix of two rows  $\{0. 1. 1. 2.2\}$  and  $\{1. 0. 1. 2.\}$ . Lines can be commented out with # sign. All the fields which do not look like numbers are skipped. If you matrix is symmetric, you may specify only the upper left or the lower right triangle like this:

1. 1. 2 1. 2 -1. 1. 2 3. 5.

## 2.23.39. Numerical data (real arrays) ( \*.rar )

File may contain arbitrarily mixed numbers and strings. Strings will be skipped and numbers will form an array. A hash sign # at the beginning of a line comments this line out.

Examples:

# 1.2 1.4 1.8 rem 2.2

This array will lead to  $\{1.4 \ 1.8 \ 2.2\}$  array.

# 3. User's guide

This section of the ICM manual contains answers to some popular questions and example scripts for different tasks. We assume that you now understand the basic syntax and concepts of the ICM-shell and elementary molecular operations.

## 3.1. ICM-shell

## 3.1.1. How to get help

This entire book can be searched from the command line. You may just type help and use */searchString* to find what you want, or use **help commands** or **help functions** to find out about the syntax. If you want help on a multi-word term, e.g. read pdb, merge the words, e.g. help readpdb. The web version of this book with links is also available (man.tar.gz file contains about 2000 html files). See also: help, help commands, help functions.

## 3.1.2. Customization

There are several mechanisms of helping you to customize your ICM environment:

#### ICMHOME

Define UNIX environmental variable ICMHOME in your .cshrc file, e.g.

setenv ICMHOME /opt/icm/

This will be the directory from which ICM program takes all necessary libraries, databases, etc.

#### USER environment: \$HOME/.icm files and directories

In contrast to \$ICMHOME directory which may be installed under root for several users, the \$HOME/.icm directory contains files which can be changed by a user. Under Windows the directory is \$USERPROFILE/.icm. The icm-user directory contains the following files and directories:

dir/file	description					
blastdb/	contains blast-formatted files for ICM sequence searches (see also s_blastdbDir)					
config/	contains icm.cfg and icm.rc files with startup preferences.					
icm_projects/	contains subdirectories with icm files for saved project (see also s_projectsDir).					
inx/	default location for database index files (see also s_inxDir)					
log/	contains session logs, (see also s_logDir)					
_startup	if present, overrides the global _startup in \$ICMHOME.					
user_startup.icm user startup files. Executed in addition to _startup						

#### general \_startup

Adjust \_startup file to your needs. Define directories, PostScript viewer, help engine, database directories, etc.

Examples are given below.

```
s_pdbDir = "/data/pdb/" # set it to the place where PDB lives
s_helpEngine = "icm" # default, web-browser is an alternative
s_prositeDat = s_icmhome+"prosite.dat" # you may have your own updated file
s_psViewer = "/usr/opt/bin/gs -q" # xpsview may be more standard
s_tempDir = "/tmp" # better be accessible only for you
```

Set \$BLASTDB system environment variable for searches in the blast-preprocessed sequence databases (see the find database command).

#### individual user\_startup.icm , gui-controls and menus :

additional **personal custom commands** can be stored in \$HOME/.icm/user\_startup.icm file under UNIX and \$USERPROFILE/.icm/user\_startup.icm under Windows. This allows to have your own additional setup which follows a general \$ICMSCRIPTDIR/\_startup[.icm] file execution. We also recommend that you create a .icm directory in your home directory and store all additional files, such as your personal icm.clr, GUI-setup and color file, and other custom files. Example personal user\_startup.icm file in which we use personal GUI files:

LIBRARY.clr = Getenv("HOME")+"/.icm/icm.clr" # USERPROFILE under Windows LIBRARY.men = Getenv("HOME")+"/.icm/icm.gui" # now gui will invoke your file read color # updates your GUI-control and color setup

#### aliases :

Define aliases convenient for you.

#### hotkeys :

Use, define new or modify old keystrokes. Set key command allows you to create efficient keystrokes to control graphics

#### macros:

Create macros for all frequently used interactive operations and their combinations.

#### 3.1.3. How to write a nice demo with menus to impress the boss

See example in the description of the menu command and the \_demo family of scripts.

#### 3.1.4. How to boost learning process while reading the ICM manual

• Start ICM, mark (copy) all the example lines from the manual with the mouse and paste them to the ICM-shell text window. Try each line as you read it! You will immediately see the result and

will be able to play around. Note: in principle you may mark several lines, but sometimes the buffer gets overflown. In this case paste lines in smaller chunks or one by one.

- Use HTML cross-references.
- Write to support@molsoft.com if you have questions or need an ICM-shrink.
- Always use TAB key while typing the command.

#### 3.1.5. How to get the list of the command words

Use list command to see the whole list of valid ICM words. If you need a list of all available ICM functions, use list function command instead.

help commands and help functions will give all the syntax lines too.

## 3.2. ICM graphics

## 3.2.1. How to learn the ICM molecular graphics in 30 seconds

To master ICM graphics you only need to know the following words: **commands:** display, undisplay, color, center, delete, connect;

nouns: wire, cpk, ball, stick, xstick, surface, skin, ribbon, label, residue, atom;

#### selection: e.g.

```
a_1.  # the first object
a_1.1  # the first molecule
a_1.1/5:10 # residues from 5 to 10
a_1.1//ca,c,n  # the backbone atoms
Sphere(a_1.1 a_2. 10.) # atoms around a_1.1 in a_2.
```

colors: white, black, blue, green, etc. (see file icm.clr).

output image formats: png, tif (default), targa, gif, rgb.

Now start from the command word and type what you need, and use controls to rotate, translate, label, zoom, mark and color.

Example:

```
read pdb "lcrn"
display ribbon
color ribbon a_/4:8 blue
display xstick a_1.1/10 green
center a_/6:12
display residue label a_/6:12
display string "Crambin" 36 red
write image rgb "crn" # use IRIX imgview to check the image
write image window=2*View(window) # hi-res picture
```

More advanced topics: connect, graphics objects.

## 3.2.2. How to make a nice high-resolution image

So called *computational biology* is primarily about generating nice pictures. Here are a few tips.

1) **image display.** Display your molecule or molecule the way you like. Running the nice macro (e.g. nice "lest") is a good start.

2) **background.** Change **background** color with Ctrl-E and Ctrl-Q to your liking.

3) **fog.** It is a great visual effect. Use it! Switch on the depth-cueing effect, i.e. fog with Ctrl-D (depth). Move the bright front clipping planes with Ctrl-MidMB and move the back clipping plane closer with vertical movements with MidMB at the right margin of your graphics window (the mapping of the mouse controls to effects is defined by the icm.clr file).

You may color the **fog** with color volume *color* command and modify fogStart to increase or decrease the unfogged slice of the molecule.

4) grobs. Make them smooth with select  $g_{...}$  and pressing Ctrl-X and unselecting with the Esc button. To shine light from outside sometimes you need to say display reverse.

5) **quality** of graphic elements. set IMAGE.quality=12 or 15. This parameter determines the number of triangles in shapes like spheres and cylinders. The default value is only 5. because at routine interactive work you prefer speed to quality.

6) **image resolution.** Let us count pixels. Your screen window is usually about 600x600 pixels which corresponds to 2x2inch picture at 300dpi resolution, or 1x1 inch at 600dpi. Therefore, if you want to generate image at high resolution use two tricks: (i) make your window as large as possible; (ii) use the write image command with option window= and select a factor 2 or 3 to generate 2 or 3 times large image respecively, e.g.

```
write image png window= 2 * View(window)
```

Be careful, the window option will not save the text labels. It is preferable to add text labels outside ICM anyway.

7) The picture is ready. Enjoy the attention.

## 3.2.3. How to rotate one molecule around its own center of mass

First, you need to move the molecule of interest to the center of the screen: center to this molecule and use the connect a\_molselection command to 'connect' your mouse to just this molecule and leave everything else unchanged. You will see that E.g.

```
read pdb "2ins"
center a_b
connect a_b  # use mouse LB to rotate and translate
```

## 3.2.4. How to annotate a molecular image in the graphics window

Simply use display string command allowing to place a string into the graphics screen. The string can be dragged later to any location with the middle mouse button. You can set font, color and positions you like. Obsolete strings can also be undisplayed or removed by the command delete label, or by the BACKSPACE key when the cursor is in the graphics window (see keyboard and mouse controls).

### 3.2.5. How to save and print the generated image

Save the image using write image command, preview it with the IRIX imgview, convert it to the PostScript format and print it. You may also save image directly in PostScript format, either as a bitmap snapshot or as a vectorized high-quality model of linear, triangular and string primitives, as they are rendered on your display. See

write image

and

```
write postscript
```

for details.

### 3.2.6. How to change the color of the graphics window background

Command color background allows to set any color from those specified in the file icm.clr, for example

color	background	white	#	or
color	background	black	#	or
color	background	aquamarine		

Note that numeric values may also be used, for example:

```
display string "background\ncolor test" 0.4, 0.9
ncolor = 127
for icolor = 1, ncolor, 1
    color background icolor
    display string "color number " + String (icolor)
    delete label 2
endfor
delete label
display string "the end"
color background black
```

See also icm.clr file.

## 3.2.7. How to return a molecule to the center of the graphics window

Use the center command.

3.2.4. How to annotate a molecular image in the graphics window

### 3.2.8. How to color atoms according to their B-factors

Command color can use any real array as a set of individual color numerical specifications. The smallest number corresponds to the red color, the largest number to the blue one. Function  $\texttt{Bfactor}(as_)$  returns the real array of B-values for each selected atom.

Examples:

```
read pdb "lcrn"
color a_lcrn.//* Bfactor(a_lcrn.//*)
```

#### 3.2.9. How to color residues according to their hydrophobicities

It is very simple:

```
read object "4pti"
display a_//!h* white
display surface a_1.1 a_1.1
s_method = "surface"
                       # s_method = "xstick" or "cpk" is also possible
                                          # hydrophobic
color $s_method yellow a_/ala,val,phe,ile,leu,pro,met/!c,n,o,hn
                                          # polar
color $s method pink
                       a_/ser,thr,tyr,cys,asn,gln,his,trp,gly
                                          # charged (+)
color $s method blue
                       a_/lys,arg/nz,hz*,nh*,hh*
                                          # charged (-)
                       a_/asp,glu/oe*,od*
color $s_method red
display string yellow
                       "hydrophobic: yellow" 25, -0.9, 0.9
                                            25, -0.9, 0.6
display string pink
                       "polar: pink"
display string blue
                       "charged(-)"
                                            25, -0.9, 0.8
display string red
                       "charged(+)"
                                            25, -0.9, 0.7
```

See also How to color atoms according to their charges.

#### 3.2.10. How to color residues according to their accessibilities

Function Area is required.

Example:

3.2.8. How to color atoms according to their B-factors

## 3.2.11. How to color atoms according to their charges

Use function Charge . For example:

read object "crn"
display surface
color a\_//\* Charge(a\_//\*)

Another example is selected residues coloring:

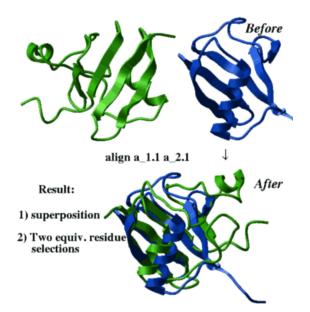
display a\_\*./asp,glu/o?\* cpk red display a\_\*./lys,arg/nz,n?\* cpk blue

## 3.3. Structure analysis

## 3.3.1. How to optimally superimpose two 3D structures

Optimal superposition implies optimization of the Ca–RMSD upon rigid body superposition of the *equivalent* residues/atoms. This set of equivalent positions can be predefined, or determined by sequence alignment, or automatically derived from structure. In the latter case the optimized value is the RMSD of a trial alignment is corrected by the alignment length to reward longer alignment with slightly worse RMSD.

There are several different algorithms which can be applied:



- superimpose by known residue or atom equivalences (see the superimpose command)
- superimpose by sequence alignment which is calculated on the fly:

superimpose a\_1.1 a\_2.1 align # the sequences are generated on the fly

This procedure fails if two structures have no significant sequence similarity

• align ... command to find *global structural* alignment (returned by the ali\_out variable) and superimpose accordingly. Here we do not rely on sequence alignment (although it can be

added to the optimized scoring function with a certain weight).

align a\_1.1 a\_2.1

• Align(.. distance) for local structural alignment

Use superimpose command. It performs an optimal rotation and translation of one structure onto the other. If necessary, a sequence alignment may be done prior to superposition by specifying align option in the command line.

Example:

Note, in this particular example, the whole structural similarity is not so high. However, better fit may be obtained if only portions of the structures are superimposed, for example:

superimpose a\_1.1/16:27/n,ca,c a\_2.1/116:127/n,ca,c align

See also: Rmsd() and Srmsd().

#### 3.3.2. How to optimally superimpose without the residue alignment

The core of this procedure is the

align ms\_molecule1 ms\_molecule2

command. There are two variants: a fast superposition using dynamic programming algorithm align [distance]  $ms_1 ms_2$  or a more rigorous, but somewhat less stable and slow align heavy  $ms_1 ms_2$  ... command. This first command is well described above and identifies only the best superposition. The initial superposition is then refined similarly to the find alignment command.

The second algorithm (option heavy) identifies a number of possible superpositions (solutions) based on the Ca atom coordinates only. The first solution is the best hit. See also load solution command.

Examples:

```
read pdb "4fxc"
read pdb "lubq"
display a_*.//ca,c,n
color molecule a_*.
align a_1.1 a_2.1
center
```

```
color red as_out
color blue as2_out
show ali_out
```

## 3.3.3. How to make a Ramachandran plot

Use macro plotRama. The macro is invoked by

plotRama rs\_selection l\_addLabel l\_addBoundaries

**Important:** if a PDB structure is analyzed, convert it first to get a proper ICM-object (true ICM-molecular object does not require prior preparation for building Ramachandran plot).

Example:

## 3.3.4. How to display hydrogen bonds

A list of hydrogen bonds may be calculated and displayed for an ICM-objects or non-ICM object with hydrogens. If you are dealing with a PDB structure without hydrogens, convert it first. The command show hbond prints list of hydrogen bonds in the text window. After that they can be displayed. (Hydrogen bonds can also be calculated by minimize and show energy commands provided that the hydrogen bond energy term is switched on.)

The display hbond command allows to show the deviation angle of the hydroben bond from linearity (see the GRAPHICS.hbondStyle preference).

Examples:

See also: GRAPHICS.hbondStyle

## 3.3.5. How to identify atoms or residues at the molecular interface

When two or more parts of the polypeptide chain(s) are near each other in space, they are referred to as a molecular interface. What "interface" is can be defined more specifically in the context of a particular

study, so here only an example is given to illustrate how interface may be identified and displayed. Two ICM functions are to be used for that, viz. Sphere and Acc. Suppose, you analyze 3D structure of a complex of two molecules, and would like to see what residues are at the interface. It can be done by the following:

```
read object "complex"
display a_1,2//!h*
                          # display both molecules
                          # without hydrogens
color a_1 yellow
color a_2 green
show area surface a_1//!h* a_1//!h* # calculate surface of
                                        # the 1st molecule only
color red Sphere(a_2//* a_1//* 4.) Acc(a_1/*)# interface residue atoms
                                            # of the 1st molecule
                                            # in 4 A radius vicinity
                                            # of the 2nd molecule
show area surface a_2//!h* a_2//!h* # calculate surface of
                                            # the 2nd molecule only
color blue Sphere(a_1//* a_2/* 4.)# interface residue atoms
                                            # of the 2nd molecule
                                            # in 4 A radius vicinity
                                            # of the 1st molecule
```

If, instead, you need to mark **residues**, convert the selection of the interface atoms to residues with the Res () command:

```
.... # same as above.
....
resAtInterf = Res( Sphere(a_2//* a_1//* 4.)) Acc(a_1/*)
display residue label resAtInterf
```

Note that in the Sphere command it does not matter if you specify the atom selection or a residue selection as an argument, since the function operates at the atom level anyway. The difference in the specification of the ICM selection in these two examples (usage of **two** slashes for atom selection, and **one** slash for residue selection):

```
Sphere (a_1.1/* 4.) versus Sphere (a_1.2//* 4.) and also Acc(a_1.1/*) versus Acc(a_1.2//*) for specifying residues and atoms, respectively.
```

**Important:** when calculating surface, be sure that you properly specify the selection arguments in the show area surface command.

#### 3.3.6. How to identify torsions at the molecular interface

Identification of the torsions belonging to residues at the molecular interface is a necessary and non-trivial step in many tasks of the molecular modeling. An example below shows how this identification may be done in ICM. The same as above ICM-object "complex" is considered.

```
read object "complex"
```

```
display a_1.1,2//!h*
                                       # display both molecules
                                       # of the complex w/o hydrogens
color a_1.1 yellow
color a_1.2 green
show area surface a_1.1//!h* a_1.1//!h*
                                           # calculate surface of
                                           # the 1st molecule only
a1=Sphere(a_{1.2}/* 7.) Acc(a_{1.1}/*) # define the 1st molecule atoms
                                       # belonging to the interface
                                       # of the 1st molecule
                                       # in 7 A radius vicinity
v1=V_1.1//S a1
                                     # identify standard geometry
                                       # torsions of the 1st molecule
                                       # belonging to the interface
color red Atom(v1)
                                       # color atoms which torsions
                                       # belong to
                                       # similar for the 2nd molecule
show area surface a_1.2//!h* a_1.2//!h*
a2=Sphere(a_{1.1}/* 7.) Acc(a_{1.2}/*)
v2=V_1.2//S a2
color blue Atom(v2)
```

### 3.3.7. How to calculate packing density

The packing density analysis requires understanding of two types of surfaces: the skin (molecular surface) and solvent-accessible surface of water probe **centers** (which is one water radius away from the skin). The following is an example of how it may be done for a fragment of a protein.

```
read object s_icmhome+"crn"
asel = a_/5:15
show volume skin asel asel
rskin = r_out
vwExpand = 0.
show volume surface asel asel
rsurf = r_out
print "skin volume = ", rskin, "; vw volume = ", rsurf
print "packing density = ", rskin/rsurf
```

## 3.3.8. How to perform a principal component analysis

For a set of objects with given measure of similarity between each two of them, one can easily perform the principal component analysis or to solve a distance geometry problem by using Disgeo function. The following example shows how to get a two-dimensional distribution of amino acid sequences of a series of Zn-fingers given the distance between sequences is defined by Distance(*sequence1 sequence2*) function. This distance is essentially a measure of sequence similarity: the distance is 0. for two identical sequences, it is proportional to percent identity divided by 100. for very similar sequences and goes above one at about 30% sequence identity, tending to infinity for very small seq. identity numbers.

```
align alZnFing
                            # align them, then ...
a = Distance(alZnFing)
                            # calculate a matrix of pairwise
                            # distances among them
n=Nof(a)
                            # number of points
                            # calculate principal components
b=Disgeo(a)
corMat = b[1:n,1:n-1]
                            # coordinate matrix [n,n-1] of n points
eigenV = b[1:n,n]
                            # vector with n sorted eigenvalues
xplot = corMat[1:n,1]
yplot = corMat[1:n,2]
                            # plot 2D distribution
plot xplot yplot CIRCLE display
```

#### 3.3.9. How to calculate a dihedral angle

Normal dihedral angles like torsion angles describing conformation are directly available through the Value() function. You need to convert an object into the ICM type if necessary.

Example:

```
read pdb "lcrn"  # read it in
convert  # quickly convert into ICM-object
show v_//phi,psi  # just show all phi,psi's
show v_/3:17/xi*  # show chi angles of residues from 3 to 17
a = Value(v_//phi,psi) # create a real array of values of spec. torsions
```

You can calculate a dihedral angle between any two planes defined by three atoms (for example two Phe rings) with the calcDihedralAngle macro. If this macro is loaded, you can do the following:

```
read object "crn"
as_1 = a_/1/n,ca,c
as_2 = a_/3/n,ca,c
display a_//n,ca,c blue
color as_1 magenta
color as_2 green
calcDihedralAngle as_1 as_2
print "dihedral angle 1(n,ca,c) and 3(n,ca,c) (deg) = ", r_out
```

Note that the order of atoms in the selections  $as_1$  and  $as_2$  is determined only by the ICM–molecular tree (will be the same for a\_//n,c or a\_//c,n). Thus, any changes in the selections  $as_1$  and  $as_2$  not changing their content has no effect on the resulting dihedral angle:

```
as_1 = a_/1/ca,c,n
as_2 = a_/3/ca,c,n
calcDihedralAngle as_1 as_2
print "dihedral angle 1(n,ca,c) and 3(n,ca,c) (deg) = ", r_out
```

See also: Acos(), Length(), Sum(), Vector().

## 3.3.10. How to print a table of the torsion angles

The simplest list can be generated by:

 show V\_//xi\* # side chain torsions

A nicer formatted output (one line per residue) may be generated with macro printTorsions, for example:

read pdb "lcrn"
convert
printTorsions a\_/2:15

Note, that you do not need to convert your molecular object if it is an ICM-object.

## 3.3.11. How to build a hydrophobicity profile

First, define a hydrophobicity scale, for example that from Kyte and Doolittle, 1982 or use your favorite one. (Note, there should be 26 entries in the hydrophobicity parameters list *hPhobInd* corresponding to the 26 letters of the alphabet. Non–participating letters B,J,O,U,X,Z are marked by zero values.)

```
# define a hydrophobicity scale
hPhobInd = { 1.8, 0.0, 2.5, -3.5, -3.5, 2.8, -0.4, \
           # make a macro
macro hPhobProfile seq_ i_windowSize
  if (Type(i_windowSize) == "unknown") then
   i_windowSize = Ask("Enter window size", windowSize)
 endif
 R_window = Rarray(i_windowSize,1./i_windowSize)
 R_hphob = Smooth (Rarray(seq_,hPhobInd), R_window)
 R_ruler = \{0., 0., 10., 10., 0., 0., 0., 0.\}
 R_ruler[2] = Real(Length(seq_))
 r_tic = 1./Sqrt(Real(i_windowSize))
 r_tic = Integer(r_tic*100.0)/100.
 R_ruler[5] = -7.*r_tic
 R_ruler[6] = 7.*r_tic
 R ruler[7] = r tic
 R_ruler[8] = r_tic
 print R ruler
 s_legend = { "Hydrophobicity plot", "Sequence", "Hydrophobicity" }
 xplot = Count(1 Length(seq_))
 yplot = R_hphob
 psfilename = "hphob"
 plot xplot yplot R_ruler s_legend grid="xy" display psfilename
 delete R_window R_ruler r_tic
endmacro
                                # now, an example
read object "crn"
s = Sequence(a)
hPhobProfile s 7
```

## 3.3.12. How to display and characterize protein cavities

ICM offers fast, elegant and mathematically accurate way to identify, display, and measure protein cavities. An example session which displays all cavities with their surroundings, and calculates their volumes and

surface areas:

Examples:

```
read object "crn" # or whatever
make grob skin "g skin"
split g_skin
nShells = i_out
display wire residue labels
for i=1,nShells
  v = Volume(q_skin$i) # actually its surface is returned in r_out
                          # there is no need to use the explicit Area(g_skin$i)
  s = r_{out}
                        # note that cavities have negative volume!
  if(v > 0.) then
    display transparent smooth g_skin$i
    printf "Shell %d: V=%f A=%f\n", i, v, s
   else
     display reverse smooth yellow g_skin$i
     center q skin$i
    printf "CAVITY %d: V=%f A=%f R~%f\n", i, -v, s, -3.*v/s
  endif
# add pause here for an interactive session
endfor
```

## 3.4. Sequence, searches and alignments

#### 3.4.1. How to search all Prosite patterns in your sequence

Use macro searchSeqProsite. For example:

read pdb "2dhf"	
make sequence a_1.1	<pre># sequence of a PDB structure</pre>
show sequence	
find prosite 2dhf_a	<pre># 2dhf_a is the sequence of the protein</pre>

See also find prosite, find pattern and read prosite.

## 3.4.2. How to find a fragment in the PDB database ( obsolete )

First, make sure that you have a library of representative icm-objects. String variable s\_qsearchDir should contain the relative path of this directory with respect to the s\_dataDir directory. The library may be created and updated with the provided \_mkQsearchLib script. Use qsearch or iqsearch macros. Load the object and type qsearch or iqsearch + arguments. You will be prompted for the forgotten arguments.

To understand the meaning of the arguments, see the find pdb command.

Examples:

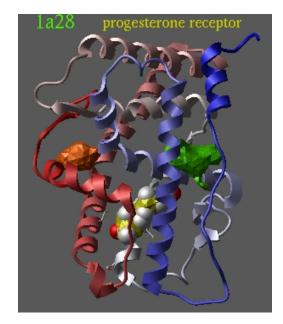
iqsearch a\_1crn./2:6,14:18 "xxxxx-----xxxxxx" "\*" "\*" .7

### 3.4.3. How to identify binding pockets

There are three algorithms (A, B, and C) with ICM which can identify pockets:

option	target	macro
А	closed pockets	icmCavityFinder
В	almost closed pockets	make map potential, etc., see below
С	pockets with good ligand-binding potential	icmPocketFinder

For the areas of space attracting ligands (option C), use two macros:



Example:

read pdb "1a28"
delete a\_!1,2
convert
delete a\_2
icmPocketFinder a\_ 3.

In the following example we find an **almost closed pocket** which can not be identified with icmCavityFinder .

read pdb "lfm6" # read the 'a' chain of RXR delete a\_!1,9 # keep the RXR and its ligand only make map potential a\_1 Box( a\_ 1. ) 1. # grid size 1.5 A make grob m\_atoms exact 0.1 solid split g\_atoms cool a\_ display g\_atoms2 reverse If you have problems with identifying pockets, change the grid size, the threshold level for make grop m\_atoms, or try to convert object to the ICM type (the conversion will add hydrogens and make the object more dense).

### 3.4.4. How to find a similar fold or topological motif in the PDB database

Use macro searchObjSegment, for example:

```
read object s_icmhome+"crn"
searchObjSegment a_1.1 30 3.
# or
read pdb "lpxt"
delete a_!1
convert
searchObjSegment a_1.1 24 6.
```

You may need to adjust the seed fragment length and the RMSD parameters for a cleaner list.

The database foldbank.seg is provided and may be recompiled, customized and updated by the supplied \_mkSegmentLib script.

See also segment, find segment, write segment, foldbank.seg, How to extract a diverse set of PDB entries How to compile a database of protein secondary structures and their folds.

## 3.4.5. How to generate a non-redundant list of PDB sequences

The following script is a skeleton of the provided script \_mkUniqPdbSeqs which is somewhat more automated.

```
l_commands=no
                           # if something goes wrong do not
errorAction="none"
                           # interrupt the loop
read sarray s_pdbDir+"/derived_data/index/source.idx"
                          # you need a list of all pdb-entries
                          # (4 char. code per line will do)
source = Tolower(Trim(Field(source,1)))
n=Nof(source)
for i=1,n
  read pdb sequence resolution source[i]
        # append resolution to the chain name (like 9lyz_a19)
endfor
group sequence "*" uniqSeqs unique 0.1
        # cutoff inter-sequence
        # distance 0.1 (dissimilar by more than 10%)
#
#
 Other possibilities
#
#
  group sequence uniqSeqs unique 5
                                         # if two seqs differ by more
#
                                         # than 5 mutations
#
  group sequence uniqSeqs unique
                                         # throw away only identical
#
                                         # sequences
#
                                       # get rid of sequences not
delete sequences
                                       # included in uniqSeqs
```

#### 3.4.6. How to merge several pdb files

The simplest way to merge two pdb files is to read them as separate objects and the use the move  $a_1$ .  $a_2$ . command. Example:

```
read pdb "lcrn"
read pdb "ld48"
move a_2. a_1.  # merges objects
write pdb a_1. "both" # saves both files in pdb format
write object a_1.  # saves merged object in compact binary form
```

Before or after merging, the objects can also be edited, translated to a new position, rename chains, change residue numbers etc. Example:

```
read pdb "1d48"
delete a_w*
delete a_2  # delete the second chain
read pdb "1crn"
delete a_/33:99  # delete a C-term. part of crambin
move a_1. a_2.  # merge the remains
write object a_
```

If you want to re-engineer a polypeptide chain of a protein, using two pdb-files, e.g. to transplant one part of a protein to another and restore the bonding connectivity, you may use the modify command:

```
read pdb "lcrn" # one pdb
read pdb "lcbn" # similar protein
modify a_1./20:25 a_2./20:25
    # translants a loop from 2nd object to the lst one
write pdb a_1. "combo"
```

# 3.4.7. How to compile a database of protein secondary structures and their folds

The following script uses the previously compiled list of unique pdb chains and creates two files: foldbank.db containing sequences, resolutions, the deposited and the automatically assigned secondary structures of the nonredundant set and foldbank.seg containing quantitative topology descriptions of the folds. The GAP (which stands for Gly-Ala-Pro) library allows to build only the backbones necessary for the secondary structure prediction algorithm and speeds up the PDB->ICM conversion. The foldbank.db is in the ICM database format, so that you can create an ICM table shell-object. This allows to sort entries and perform searches to create subsets.

```
l_commands =no
l_info =no
l_confirm =no
errorAction="none"
segMinLength =3
mncalls =300
```

3.4.6. How to merge several pdb files

```
="./"
s_icmhome
s reslib
             ="icmGAP" # Gly-Ala-Pro residue library
read library
      # ...getting the representative list of chains ...
read sequences s_pdbDir+"/derived_data/pdb_seqres.txt"
      #make sure to have mkUnigPdbSeqs executed recently
li=Name(sequence)
delete sequences
      #...you may modify the method or create your own list...
if (Error) quit
unix my foldbank.db foldbank.db.OLD
unix mv foldbank.seg foldbank.seg.OLD
for i=1,Nof(li)
 lii=Tolower(li[i])
 read pdb lii[1:4]+"."+lii[6]+"/"
                                        # delete HET-molecules
 delete !Mol(a_*/A)
 convert
 er=r out
 rz=Resolution(a 1.)
 if(rz < 0.01)rz=9.99
 sx=Sstructure(a_*)
 assign sstructure
                     # uncomment the following line, if you'd like
                     # to save GAP objects. requires GAP subdirectory
                     # write object "GAP/"+lii[1:4]+lii[6]
 sprintf "# %d\nNA %s.%s\nRZ %.2f\nER %.3f\nSE %s\nSX %s\nSS %s\n" \
           i lii[1:4] lii[6] rz er String(Sequence(a_*)) sx Sstructure(a_*)
 write append s_out "foldbank.db"
 assign sstructure segment
 rename a_2. lii[1:4] # restore the original pdb-name
 write append segment "foldbank.seg"
 delete a *.
endfor
quit
```

#### 3.4.8. How to search headers of the PDB entries

There is an PDB.tab file which contains one line header descriptions of all the entries. Now you have three ways of doing it:

• In unix:

grep -i kinase PDB.tab

• From ICM: you have more possibilities:

```
read table s_icmhome+"PDB.tab"
show PDB.head ~ "kinase" # or
show PDB.comp ~ "kinase*" # or
show PDB.comp ~ "kinase*" # regular expressions
# You can also
a=PDB.head~"kinase"
for i=1,Nof(a)
    nice PDB.ID
    pause
    delete a_*.
```

endfor

• Use the gui (Find.In PDB.By Keyword..)

## 3.5. Energetics and electrostatics

## 3.5.1. How to plot the distance dependence of a van der Waals interaction

The following script will plot three energy-interatomic distance plots for three possible van der Waals terms defined by the vwMethod preference ("exact"-black,"soft"-blue and "old soft"-red). Simply mark and paste the following lines into your ICM session:

```
buildpep "one;one" # two oxygens
set term "vw" only
set a_2//o Sum(Xyz(a_1//o ))+{.0 .0001 .0}
n=200
a=Rarray(n)
b=a
c=a
r=Rarray( n .03 3.)
vwSoftMaxEnergy = 14.5
for i=1.n
   translate a_2 add {0.03 0. 0.}
#
   vwMethod="exact"
   r[i]=Distance( a_2//o a_1//o ) # use Sum(Distance(..)) for the old version
   show ey mute
   a[i]=Energy("vw")
#
   vwMethod="old soft"
   show ev mute
  b[i]=Energy("vw")
   vwMethod="soft"
   show ey mute
   c[i]=Energy("vw")
endfor
s=Sarray(3*n)
s[n+1]="_red line"
s[2*n+1]="_blue line"
plot ds r//r//r a//b//c s {0. 6.01 1. 1. ,-1. 17. 1. 1.}
```

## 3.5.2. How to calculate the electrostatic free energy by the REBEL-method

This short script solves the Poisson equation by the "Rapid-Exact- Boundary ELement (REBEL) method for crambin.

Examples:

```
electroMethod="boundary element"
read object "crn"
delete a_w*  # get rid of water molecules
show energy "el"
show Energy("el")-r_out, r_out # Coulomb and solvation components
```

## 3.5.3. How to evaluate the pK shift

Suppose we mutate a surface Asp into Ala and want to evaluate how the pK of the neighboring His is changed. The pK shift may be evaluated as the difference of potential at Nd1 His and Ne2 His nitrogens due to the mutation. Since Ala may be considered as uncharged, the shift is simply the potential at the nitrogens due to the Asp charge.

Example (pKshift of His34 from Asp22):

## 3.5.4. How to evaluate the binding energy

There are many different approaches to the evaluation of binding energy. One of the reasonable approximations has the following features:

- van der Waals/hydrogen bonding interaction is excluded since it has close magnitudes for protein-protein and for protein-solvent interactions;
- electrostatic free energy change is calculated by the REBEL method (see also the section "How to calculate the electrostatic free energy ... ") above);
- side-chain entropy change is calculated by standard ICM entropic term based on exposed surface area of flexible side-chains;
- hydrophobic energy change is calculated using surface term with constant surface tension of 20. cal/Angstrom.

Example:

```
electroMethod="boundary element"
surfaceMethod="constant tension"
surfaceTension=0.020
dielConst = 12.7
set terms "sf,el,en"
read object s_icmhome+"2ptc"
show energy a_1 a_1 mute
el =Energy("el,sf,en")
show energy a_2 a_2 mute
e2 =Energy("el,sf,en")
show energy mute
e12 =Energy("el,sf,en")
print "Binding energy = ", el2 - el - e2
```

## 3.5.5. How to calculate an ensemble average

The following is an example of calculating the average of an interatomic distance over a set of conformations collected in the conformational stack. This calculation is written as a macro. Feel free to change it. You may also use movie and load frame instead of stack and load conf, respectively.

```
# first, define the macro
macro ensembleAverage r_temperature
l_commands = no
l_{info} = no
load conf 0
                                 # extract the lowest energy
 e0 = Energy("func")
ansAver = 0.
                                  # the statistical sum initialization
statsum = 0.
r_temperature = r_temperature * Boltzmann
for i = 1, Nof(conf)
                                  # loop through all the stack
                                  # conformations
  load conf i
  prob = Exp((e0-Energy("func"))/r_temperature)
                                  # averaging distance between two ca
                                  # is just an example
   ansAver = ansAver + Distance(a_2/2/ca a_4/ca)*prob
   statsum = statsum+ prob
endfor
r_out = ansAver/statsum
print " Ensemble average is: ", r_out
endmacro
                               # Now you can calculate your average
read object "my_peptide"
read stack
                              # stack file is assumed to have
                              # the same name
ensembleAverage 600.
                              # sometimes you may use the elevated
                              # temperature to account for relaxation
```

#### 3.5.6. How to evaluate helicity of a peptide from the BPMC simulation

1. Run the \_folding script first. Make sure the procedure converges by running several simulations (say \_f1 \_f2 \_f3) from different random starting conformations. E.g.:

cp \$ICMHOME/\_folding \_f1 # adjust the script icm \_f1 > f1.ou cp \_f1 \_f2 icm \_f2 > f2.ou cp \_f2 \_f3 icm \_f3 > f3.ou

- 2. You can evaluate helicity for each simulation. If they converge the result will be about the same.
- 3. Helicity is just the ensemble average of the parameter which can be calculated as the relative number of the helical residues. Therefore you need to assign secondary structure for a particular movie frame or stack conformation and count number of helical residues. See macro \_helicity averaging helicity over the movie frames.

```
macro helicity s_movieName r_temperature
# attention: 'temperature' is extremely important.
# You may use elevated temperature to account for relaxation.
l_commands=no
l_info=no
read conf 0 s_movieName
```

```
e0=Energy("func") # the lowest energy
av=0.
ssum = 0.
r_temperature = r_temperature * Boltzmann
res = Real(Nof(a_/*))
read movie s_movieName
for i=1,Nof(frame)
    load frame i s_movieName
    assign sstructure
    prob = Exp((e0-Energy("func"))/r_temperature)
    av = av + prob*Nof(Sstructure(a_1/*),"H")/res
    ssum= ssum+prob
endfor
print " The best E=", e0, " Helicity= " av*100./ssum
endmacro
```

#### 3.5.7. How to merge and compress several conformational stacks

You may run several montecarlo simulations and accumulate several conformational stacks (\*.cnf files). To unite them it is essential that they have been created with the same energy function, because the compression algorithm takes the energy into account to decide which structure is more valuable. If it is not the case, you can always recalculate energies for the stack conformations by the following procedure:

```
read object "f"
read stack "f1"
read stack "f2" append # the second stack will be appended
for i=1,Nof(conf)
    load conf i
    show energy s_correctTerms # say, "vw,14,to,el,sf,en"
    store conf i
endfor
```

Now, to unite all the stacks and compress them you may do the following (just the idea):

```
read object "f"
delete stack
read stack "f1"
                     # first simulation
read stack "f2" append # second simulation
read stack "f3" append # third simulation
                      # look at what you have now
show stack
compare v_{-}/phi, psi # use the comparison criterion from the simulation script
                     # compose a new one, see also the compare command
vicinity = 40.
                    # you may redefine the vicinity parameter
compress stack
show stack
                     # look at the compressed stack
write stack "f4"
                     # save the result
```

## 3.6. Manipulations with molecules

#### 3.6.1. How to build new object from a sequence

The easiest way to build an object with one of several peptides is to use the buildpep macro. There you can use a one letter code (upper case characters) or three-letter code and separate sequences of different chains by a semicolon. Examples:

buildpep "AAAAA" # penta-alanine buildpep "ASFHGD;EQWR" # two chains

To create a DNA duplex or a compound, use GUI.

For a more flexible building procedure, follow the following steps:

1. Create the ICM sequence file either manually or using IcmSequence function, e.g.:

write IcmSequence( "FAASVRES", "nh3+", "coo-") "file.se"
read sequence "memb.seq" # creates the ICM sequence memb
write IcmSequence( memb , "nh3+", "coo-") "myseq.se"

build "myseq"

You can also build sequence directly without creating a file with the build string. See the build command and IcmSequence() function.

2. Change the default conformation using the available information. Possibilities:

- You have a template pdb file with all the coordinates. Use regul macro.
- ◆ Set particular dihedral angles with the set vs\_variableSelection R\_arrayOfValues or

set *vs\_variableSelection r\_theValue* command.

• you can use minimize, montecarlo and ssearch commands to find a low-energy conformation.

Examples:

```
build string "se nh3+ ala his leu trp coo-" set v_/3/xi1 -60. minimize
```

#### 3.6.2. How to quickly convert a pdb file into an ICM-object

Sometimes it is necessary to have a PDB file in the form of an ICM molecular object. For example, it's a convenient way to list and/or to change a torsion angle (or a series of them). All what you need is to use convert command. One more ICM-format object will be created (use show object command to see the list of currently loaded molecular objects). The above method is good only for a limited set of tasks mostly related to structure analysis. If you want to perform further conformational sampling by energy optimization it is better to regularize the pdb-object (see the next section) See also strip.

# 3.6.3. How to prepare a PDB structure for energy calculations (regularization)

Regularization is a sophisticated multi-step procedure. It consists of the following six steps.

- Preparation of a file with the amino acid sequence.
- Creating full-atom ICM-model (geometrical approximation).
- Rotational positioning of methyl groups.
- Iterative optimization of geometry and energy of the whole structure.

- Adjustment of polar hydrogen positions.
- Free minimization to check the consistency of the resulting structure.

See macro regul.

## 3.6.4. How to create a new molecule or a residue for the ICM residue library

Let us assume that your input is a pdb file with a new small organic molecule and you want to create a proper ICM object from this molecule. What is currently missing in the description may be the following:

- proper chemical bond types (single, double, triple, aromatic). To see them press Ctrl-W to switch to wireStyle="chemistry".
- hydrogen atoms (you need proper bond types to add hydrogens)
- proper atom types
- partial atomic charges
- directed graph (ICM-tree) imposed on all the atoms. This graph defines torsion angles and will be built by the write library command. In an ICM object the graph can be displayed if wireStyle="tree".

Identify the molecule or residue you would like to transform into the ICM–residue entry. To build all the above descriptions, perform the following procedures:

• read pdb "ligand" # molecule has no hydrogens

display

• set bond type 1  $a_{//*}$  # initialize all bond types by 1, the default is 0

# 0 type means that the convert command will try to guess the type

• set bond type 2 a\_//o2 a\_//p1 # type 'set bond type 2' and Ctrl-rightClick two atoms

# set all other non single bonds: 1-single, 2-double, 3-triple, 4 aromatic.

- set bond type 4 a\_//c1,c2,c3,c4,c5,c6 # an example of setting aromatic type for a ring
- build hydrogen # use 'delete hydrogen as\_' if things go wrong
- set type mmff
- set charge mmff write library command to save the residue entry file or append it to the icm.res or create you own library file.

## 3.6.5. How to modify an ICM-object

Methylation, hydroxylation, glycosylation, sulfation, amidation, phosphorylation, disulfide bond formation, peptide cyclization/bond,

ICM allows to perform most of the common chemical modifications of peptides and other biological molecules. It is easy to build a linear chain of amino acids and add N– and C– termini. D–amino acids can be introduced by adding capital D in front of the residue name (i.e. Dala). To make further modifications

we will use the modify and the make [disulfide | peptide] bond commands. Let us consider the main categories, using the nh3-DCSTVYHCK-coo peptide as an example. Start you session with

build string "se nh3+ asp cys ser thr val tyr his cys lys gly coo-"

Now, if you like to see the results of your operations, display the molecule and do the following:

- type **modify** in the command window;
- Ctrl-RightClick on the atom of interest (the selection will appear in the command window);
- and, finally, enter the quoted chemical group name and press Enter.

The popular modifications:

• Methylation:

...-NH-...+CH<sub>3</sub>->...-N-CH<sub>3</sub>+H

modify a\_/val/hn "ch3"

• Hydroxylation:

```
...R-CH<sub>2</sub>... + OH ->...R-CHOH... + H
```

where **R** belongs to side–chain of Lys or Pro.

Examples:

# 5-hydroxylysine (Hyl) in collagen
modify a\_/lys/hd2 "oh"
 # 4-hydroxyproline (Hyp) in collagen
modify a\_/pro/hg2 "oh"

• Glycosylation:

1. O–glycosylation:

...-R-OH + O-CH- Carb ->...-R-O-CH- Carb + OH

where **R** is a side–chain radical of Ser or Thr and *Carb* is an O–capped carbohydrate. Groups available for O–glycosylation are "acgl", "xyl", "agal", "bgal". You can further modify these groups.

Examples:

modify a\_/ser/og "acgl" # beta-D-N-acetylglucosaminide
modify a\_/thr/og1 "xyl" # xylose

2. N-glycosylation:

...-R-NH<sub>2</sub> + O-CH- Carb ->...-R-NH-CH- Carb + OH

where  $\mathbf{R}$  is a side–chain radical of Asn. *Carb* is an O–capped carbohydrate (see O–glycosylation above). The following example illustrates an alternative way of modification when a part of the attached group is disregarded.

If glycosylation follows hydroxylation, you explicitly do the same by N-glycosylation:

modify a\_/lys/hd2 "oh"
modify a\_/lys/o\_a "bgal" # o\_a is the new unique name for the oxygen

Alternatively (and preferably) replace hydrogen directly:

modify a\_/lys/hd2 "bgal"

• Sulfation:

 $\dots$ -R-OH + SO<sub>4</sub> -> $\dots$ -R-O-SO<sub>3</sub> + OH where R belongs to Tyr.

modify a\_/tyr/oh "sul" # tyrosine-O-sulfate in fibrinogen

• Amidation of the C-terminal glycine:

Build the peptide with the last gly replaced by the conh C-terminal residue. Tether it to the previous object and minimize tethers.

• Phosphorylation:

...-R-OH + O-PO<sub>2</sub>-OH ->...-R-O-PO<sub>2</sub>-OH + OH where R belongs to Ser, Thr, Tyr

or

...-R-H + O-PO<sub>2</sub>-OH ->...-R-O-PO<sub>2</sub>-OH + H where R belongs to Lys, His

or

 $\dots$ -R-O + O-PO<sub>2</sub>-OH -> $\dots$ -R-O-PO<sub>2</sub>-OH + O where R belongs to Asp.

```
modify a_/ser/og "pho" # skip if you have already modified this residue
modify a_/thr/og1 "pho"
modify a_/tyr/oh "pho"
modify a_/lys/hz2 "pho"
modify a_/his/hd1 "pho"
modify a_/asp/od2 "pho"
```

#### • Disulfide bond formation:

...(cys)-S-H + H-S-(cys)... ->...(cyss)-S-S-(cyss)... + H<sub>2</sub>

(note that names of the residues are changed upon bond formation (see disulfide bond).

```
#ds extended ICM model of the sequence
display
    # set only one SS-bond, disregard all previous
make disulfide bond a_/3 a_/9 only
    # MC search for plausible conformations
montecarlo
```

#### • Peptide cyclization and peptide bond:

```
...COO + NH<sub>3</sub>... ->...-CO-NH-... + H<sub>2</sub>O
```

```
build string "se nh3+ gly gly gly gly his coo-"
display
make peptide bond a_/nh3*/n a_/his/c  # form a cyclic peptide
display drestraint
minimize "ss"
minimize "ss"
```

The following example shows how to build a cyclic peptide cyclosporin A:

```
#
  read pdb "lcsa"
  make bond a_1csa.m/1/n a_1csa.m/11/c
#
# write library "cs" a_/1
# display grey
build string "se thr thr gly leu val leu ala Dala leu leu val"
modify a_/2/og1 a_/2/hb
modify a_/3/hn "ch3"
modify a_/4/hn "ch3"
modify a_/6/hn "ch3"
modify a_/9/hn "ch3"
modify a_/10/hn "ch3"
modify a_/11/hn "ch3"
rename a_/1 "bmt" # actually, the residue BMT is more complex
rename a_/2
             "aba"
rename a_/3 "sar"
rename a_/4 "mle"
rename a_/6
              "mle"
rename a_/9
              "mle"
rename a_/10 "mle"
rename a_/11 "mva"
display
make peptide bond a_/11/c a_/1/n
minimize "vw,14,to,hb,el,ss"
montecarlo "vw,14,to,hb,el,ss"
```

## 3.6.6. How to merge two ICM-objects

(the **move** command). It may be necessary to **merge** two or several ICM–objects or molecules to one, For example, if you are dealing with a docking problem and have prepared two molecular objects separately.

The ICM command move allows you to do that. Technically, it rearranges virtual connections in the ICM molecular tree responsible for the description of the molecules in one ICM-object or in several ones.

```
read object "complex"
                         # load a two-molecule ICM-object
display virtual a_//!h* # display molecules with virtual bonds
color molecule
show object
                         # one ICM-object loaded
read object "crn"
                        # load one more ICM-object
display virtual
color a_2. magenta
show object
                         # two ICM-objects loaded
move a_2.* a_1.
                         # merge two ICM-objects to one
                         # with virtuals connected to the origin
show object
                         # now two loaded ICM-objects becomes one
connect a 1.3
                         # you can move newly incorporated molecule
                         # w/respect to the original complex.
                         # do not forget to press ESC key in the
                         # graphics window to complete the command
                         # and / or you can save the new
                         # three-molecule object to a new file
write object "super_complex"
```

(See connect to learn more about the command.)

If, on the contrary, you would like to have one or several molecules from an ICM-object as an independent ICM-object, you should simply delete unnecessary molecules and to save the remaining one(s) as a new ICM object, for example:

## 3.6.7. How to make a hybrid model from several pdb files

To swap parts between several pdb files, read all of them to icm, and rename the chain which are you going to graft into the template, so that the template and the graft have the same name. Sometimes the two structures need to be superimposed. So, what is important for 'graftability' is

- 1. the graft has the same chain name (see rename)
- 2. the graft has residue numbers consistent with the template (see align number)
- 3. the graft has consistent coordinates (see superimpose)

Example:

read pdb "lcrn" read pdb "lcbn"

```
rename a_2.1 "m"
    # or rename a_2.1 Name(a_1.1)[1] to do it automatically
superimpose a_1.1 a_2.1 align # see more specific
```

The second concern is residue numbers. They need to be unique. This can be performed with the align number command, e.g.

align number a\_2.1/21:28 22 # renumber the loop starting from 22

Now you can write the pieces to a file and after you read it back the pieces will become one molecule.

write pdb a\_1.1/1:20 "hyb"
write pdb a\_2.1/21:28 "hyb" append
write pdb a\_1.1/29:99 "hyb" append
read pdb "hyb" # read the hybrid in
cool a\_ # display it

These operations are combined in the mergePdb macro, e.g.:

mergePdb a\_2./20:25 a\_1./300:308 # creates hybrid.pdb file

# 3.6.8. How to generate a series of intermediates between the two given structures

The following procedure will solve the problem:

```
read pdb "bj1bb"
                        # first structure
read pdb "bj2bb"
                        # second structure
                       # matrix (3, N_of_atoms) of the first ...
strt = Xyz(a_1.//*)
                     # matrix (3, N_of_atoms
# ... and of the second
fnsh = Xyz(a_2.//*)
display a_1. red
                      # to see what is going on if you need it
display a_2. blue
                        # to generate 300 intermediate conformations
nn = 300
x = 1./nn
for i = 1, nn
  set a_1.//* strt*(1.-x*i) + fnsh*x*i
  write pdb a_1. "x"+i
                 # uncomment the above line if you need
                # to save intermediates in x1.pdb, x2.pdb, etc.
endfor
quit
```

#### 3.6.9. How to reconstruct a structure from a published stereo picture

Follow these steps:

- Scan the picture and create arrays of arbitrarily scaled coordinates xLeft xRight and Y for the Ca atoms.
- When you have the coordinates in your ICM session call the makePdbFromStereo macro.
- mark the PDB-formatted lines and paste it after the read pdb unix cat command.
- inspect the results, possibly return to step 1 and correct the coordinates or use stereoAngle = -6.
- to build all-atom model, create sequence file and use the macro regul.

Example:

```
read column "xxy" # 3 numbers in each line + a header: #> xl xr y
makePdbFromStereo xl xr y 6.
read pdb unix cat
ATOM ....
ATOM ....
# Ctrl-D
display
```

# 3.7. Animation

# 3.7.1. How to rotate and zoom in a script

The simplest script will use the View(v1, v2, r) function to interpolate between views with different rotation and zoom. Example:

```
buildpep "ACDF" # tetrapeptide
  # zoom out and create a small remote view
  vl=View() # the lst view
  # zoom in and rotate
  v2=View() # the last view
  for i=1,100
    set view View(v1,v2,i*0.01) # interpolation
endfor
```

Now you can add a specific display ( cpk, skin or anything else) and add the write image command after -- set view.

The following script and macro make a .mov file for a specified molecule.

```
# makemovie -f qt -o tamox.mov -c qt_anim -r 15 `ls -tr ../f*.tif`
print "DO NOT FORGET icmt -24; set view 10 10 640 480"
macro molkino i_tZoom (20) i_tRotate (20) l_preview (yes) R_startView R_endView s_Files ("/ic
 GRAPHICS.ballQuality = 15
 nfr=30
 for i=0,i_tZoom*nfr
   set view View(R_startView ,R_endView, 0.5-0.5*Cos(180.*i/Real(i_tZoom*nfr)))
   if(!l_preview) write image s_Files + i
 endfor
 B = 0.
 for i=0,i_tRotate*nfr
   A = 360./Real(nfr*i_tRotate) * i
   A = 180. - 180.*Cos(A/2.)
   rotate view Rot(\{0, 1, 0, \} A-B)
   B = A
   if(!l_preview) write image s_Files + (i_tZoom*nfr+ i)
 endfor
endmacro
#read object "tamoxifen"
read object "propecia.ob"
set window 10 10 640 480
GRAPHICS.ballRadius=0.4
display xstick
color a_//c* green
color a_//h* white
```

```
read rarray "v0"
read rarray "v1"
set view v0
```

## 3.7.2. How to make a molecular movie from a Monte Carlo trajectory

We assume that you have a Monte Carlo trajectory saved as an ICM movie file. Displaying this movie interactively on your graphical screen is the simplest form of animation. You just load the object, the ICM movie and run the display movie command. Adjust your window and view and the allowed types of representation. This method will not work in two situations.

- First, skin and surface are not automatically recalculated and redisplayed upon the conformational change.
- Second, if you would like to display computationally heavy forms of graphical representation the interactive movie may become too slow.

In these two situations you may resort to external tools generating movie from individual image files.

In this case, the process of making a molecular movie animating the trajectory can be split into two steps.

The first step is a preparation of a series of images stored as separate disk files. Watch the ICM movie by the display movie command and select a range of frames you are going to include to the movie. There are two ways how a series of the molecular graphics images can be saved as tif or rgb formatted files.

#### Image collection (type I)

The following molecular representations are allowed in this case: wire, cpk, ball, stick, xstick and ribbon. The image option of the ICM display movie command is required. An example:

```
build
             "alpha"
read movie "alpha"
display ribbon
color ribbon a_/1:9
                      magenta
color ribbon a_/10:18 green
print "Adjust the view, press ENTER to continue"
                   # Pause to adjust a view of the molecule
pause
                   # start of the collection of the image disk files
                   # (note, tif files are to be saved)
nframe = Nof(frame)
print "Total number of frames = ", nframe
                   # specify the range you need, for example:
ifrom = 10
ito = 20
                   # otherwise, for all frames:
 ifrom = 1
#
#
  ito = nframe
display movie ifrom ito image
```

Computationally more expensive skin and surface molecular representations (or a combination of any of these two with those mentioned above) require an ICM script, as in the following example.

#### Image collection (type II)

```
"alpha"
build
            "alpha"
read movie
display xstick a_//n,ca,c
set window 600 30 640 480 # good to convert it later into NTSC format
# change
         640 480 to appropriate dimensions for PAL
lineWidth = 3.
print "Adjust the view, press ENTER to continue"
pause
nframe = Nof(frame)
print "Total number of frames = ", nframe
ifrom = 10
ito
     = 20
icount = 0
l_confirm = no
IMAGE.compress = yes
for iframe = ifrom, ito
  load frame iframe
  icount = icount + 1
  imgname = s_tempDir + "alpha_" + String(icount-1)
  display xstick a //n,ca,c green
  display surface a_/7:12/!n,ca,c a_/7:12/!n,ca,c magenta
  write image imgname
  undisplay surface
                          # should be removed and redisplayed at the next step
endfor
```

Whatever protocol you make use of, a series of files should appear in the s\_tempDir directory. List their names in a file, for example, **alpha.lst:** 

/usr/tmp/alpha\_0.tif /usr/tmp/alpha\_1.tif /usr/tmp/alpha\_2.tif /usr/tmp/alpha\_3.tif /usr/tmp/alpha\_4.tif /usr/tmp/alpha\_5.tif /usr/tmp/alpha\_6.tif /usr/tmp/alpha\_8.tif /usr/tmp/alpha\_9.tif

(Note, image numbers are started from zero, and "\_" (underscore) is inserted before the current frame number.) Now, you can put these image files into a movie file. We suggest to use the **makemovie** command for Silicon Graphics machines. The following is a sample UNIX-shell command line which should be entered to make a non-interlaced MPEG-formatted movie with the frame size 640 by 480 saved in the Apple QuickTime movie file **alpha.qt**:

makemovie -f qt -o alpha.qt -c qt\_anim -s 640,480 -r 30 ` cat alpha.lst`

Upon completion, use the **movieplayer** IRIX command to see the movie. UNIX **man** on-line help should allow you to learn more about these (and other) helpful commands. And let us give you several **helpful hints:** 

- While the collection of the image disk files is in progress, do not hide any portion or the whole area of the graphics window by opening another window which can cover the first one, and do not move the mouse cursor through the graphics window area. This procedure typically requires quite a time, so be ready to be off your terminal/workstation for the whole time required for the image collection to complete.
- Do not forget to deactivate your screen saver before you start!
- Make sure you have enough disk storage. All images are assumed to be stored in the directory specified by s\_tempDir string variable. The easiest way to estimate the required disk storage is to store the content of a single image by the write image command, and multiply the size of the resulting tif or rgb file by the number of frames. You could do it without leaving your ICM-session. For example, if you have a molecular image in the graphics window, then:

(Note that in the above example the file is stored in the current directory, not in that defined by the s\_tempDir variable).

# 3.8. Transformations and symmetry

This section describes geometrical transformations of molecular objects and manipulations with crystallographic symmetry.

## 3.8.1. Main concepts and functions

Molecular objects and 3D density maps may contain symmetry information. This information allows to generate symmetry related parts of the density or molecular objects.

An elementary space transformation is defined by a rarray where values  $\{a1,a2,...,a12\}$  define 3x3 rotation matrix and translation vector  $\{a4,a8,a12\}$ . The complete augmented affine 4x4 transformation matrix in direct space can be presented as:

al	a2	a3	a4
a5	aб	a7	a8
a9	a10	a11	a12
0.	Ο.	0.	1.

The related commands and functions (transformation vector will be referred to as  $R_tv$ ):

command/function	description
Axis ( $R_tv$ )	calculates the rotation axis $R_3$ of the transformation. Rotation angle is returned in $r_{out}$ .

Augment (R_tv)	converts 12–membered transformation vector into the augmented transformation matrix 4x4.
Cell(os_)	returns {a,b,c,alpha,beta,gamma} of the unit cell
Rmsd( <i>as_1 as_2</i> [exact])	returns <i>R_tv</i> in R_out
Rot ( <i>R_tv</i> )	extracts the 3x3 rotation matrix
<pre>superimpose as_1 as_2</pre>	returns <i>R_tv</i> in R_out
<pre>Symgroup( i_spaceGroupNumber)</pre>	returns a chain (R_[1:12*n]) of all n transformation vectors composing the specified space group.
Symgroup (s_groupName)	returns the <i>i_spaceGroupNumber</i> from which the transformations can be determined.
transform ms_R_tv	applies transformation to an object.
Trans( <i>R_tv</i> )	extracts the translation 3-vector which is applied after rotation

# 3.8.2. How to generate symmetry related molecules

There are three steps:

- define symmetry
- find elementary transformation
- apply the transformation with the transform command.

Objects read with the read pdb or read csd commands grab the symmetry information from the files. Otherwise assign the symmetry with the set symmetry os\_ s\_spaceGroupName R\_6cell i\_NofMolecules command. This way you may define your own set of symmetry transformations. Finally, loop through all the transformations and apply the transform command.

Example: see example in the transform command or paste the following lines into your ICM session:

# 3.8.3. How to find and display rotation/screw transformation axis

Steps:

- find the transformation with the Rmsd( *as\_sub1 as\_sub2* exact) function. It returns the transformation vector in the R\_out system variable.
- find the axis with the  $Axis(R_out)$  function. This function also return the a point at the axis in R\_out and the rotation angle in r\_out
- display the axis with the Grob( "arrow",  $R_6$ ) function.

Examples:

```
read obj "crn" # let us display an axis of the alpha helix 7:17
ds a /7:17
R1=Mean(Xyz(a_/7:17/ca)) # this point will be projected onto the axis
show \text{Rmsd}(a_7:16/\text{ca} a_8:17/\text{ca} exact) \# find the transformation
vv=R_out
aa=Axis(vv)
print "The rotation angle is ",r out
gg=Grob("arrow",aa )*10.+ Sum((R1-R_out)*aa)*aa+R_out # projection of R1 on aa
display gg
center
pause
#Symmetry of an extended fragment
delete object
R1=Mean(Xyz(a_/*/ca)) # this point will be projected onto the axis
show \text{Rmsd}(a_{2:12}/\text{ca} a_{1:11}/\text{ca} exact)
vv=R out
aa=Axis(vv)
print "The rotation angle is ",r out
gg=Grob("arrow",aa )*10.+ Sum((R1-R_out)*aa)*aa+R_out # projection of R1 on aa
display gg
center
```

## 3.8.4. How to combine several transformations

To combine several transformations simply multiply their augmented matrices.

Examples:

## 3.8.5. How to build a helix from the two contacting monomers

Two steps:

- find the transformation from the two coordinate sets
- apply the transformation as many times as you need

The transformation may be helical or (a degenerate case) a rotational.

Example:

```
read pdb s_icmhome + "chq" # download two molecules related by a rot. symmetry
display a_*.
R1=Mean(Xyz(a_/*/ca))
                          # this point will be projected onto the axis
show Rmsd( a_2 a_1 exact) # we assume that molecules are the same
tv = R_out
                          # this is the transformation 12-vector
ha = r_out
                          # rotation angle (just FYI)
                          # we will regenerate the 2nd molecule anyway
delete a_2
rename a_1. "s1"
for i=1,4
                          # build a helix with 4 new elements
  j=i+1
 copy a_s$i. "s"+j
                          # names for a new subunits: s2,s3,s4,s5,s6
 transform a_s$j. tv
                         # apply tv to the last copied object
endfor
aa=Axis(tv)
print "The rotation angle is ",r_out
g_ax=Grob("arrow",aa )*10.+ Sum((R1-R_out)*aa)*aa+R_out
                           # projection of R1 on aa
display a_*. g_ax
color a_*. Count(Nof(a_*.))
```

# 3.9. Maps and factors

# 3.9.1. How to manipulate with structure factors

The basic description of structure factors is given in the Glossary.

# 3.9.2. How to calculate phases of reflections given a 3D model and a cell

Basic steps:

- define crystallographic cell, map gridstep, and resolution (i.e. max h,k and l)
- build electron density map with the specified parameters
- calculate structure factors

Example:

In order to compare calculated phases I would recommend using the same factor table when calculating the \_second\_ set of phases. Then "ac" and "bc" columns will be simply replaced by the new calculated data, and new "phase2" column would be right next to the "phase" column, relating to the original object. In

addition to the previous script:

```
read pdb "ligd"
                                     # use your PDB name here
myCell
       = Cell(a_1.)
gridstep = 0.5
maxHKL = Iarray(0.45 * myCell[1:3]/gridstep)
defSymGroup = 19 # Symgroup("P 21 21 21")
make map "myMap" gridstep
make factor maxHKL "mySF"
phase = Atan2(mySF.bc , mySF.ac)
                             # ... now the 2nd object ...
read pdb "lcrn"
                             # the second PDB, not 1crn, of course
superimpose a_1. a_2.
  # you may need an alignment, see superimpose
make map "myMap" gridstep myCell
                                   # to have the same cell!
make factor maxHKL "mySF"
phase2 = Atan2(mySF.bc , mySF.ac)
delta = Remainder(phase2-phase , 360.0)
group table mySF phase phase2 delta # maybe something else
show mySF
```

# 3.9.3. How to automatically place a fragment into density

This section needs to be written. Make maps, set GRID.margin to about the size of the ligand and run montecarlo.

# 3.10. How to plot

# 3.10.1. How to make a simple plot y=F(x)

If your function is determined on an uneven set of abscissa values, stored in array x, then use:

plot display x y

If your function is determined on a regular 1D–grid of values varying from x0 to x1, and is stored in the array y, use:

plot display Rarray(Nof(y), x0, x1) y

Use on-line help:

help plot

or to see the concise list of arguments

help command plot

## 3.10.2. How to plot a histogram

Function Histogram returns the 2\*N matrix which may be used on the fly with a command like

```
plot display BAR red Histogram(a 50)
```

3.9.3. How to automatically place a fragment into density

More examples:

```
a=Random(0. 100. 10000)
b=.04*(Count(1 50)*Count(1 50))
annot = {"Histogram with quadratic ruler" "Random value" "N"}
plot display BAR red Histogram(a b) annot
b=Sqrt(100.*Count(1 100))
annot = {"Histogram with square root ruler" "Random value" "N"}
plot display green BAR Histogram(a b) annot
```

# 3.10.3. How to make a 3D-surface plot of a 2D-function

In the most common case you may have a matrix  $N*M my_matrix$  representing a two-dimensional function of x and y real arrays. First create a graphics object, then display it and save as a picture if you wish:

## 3.10.4. How to create a new graphics object of a specific shape

First, use one of the provided graphics objects, stored in \*.gro files to try these kind of ICM non-molecular objects. You can scale graphics objects (by multiplicating them by a real number) and/or adjust their positions either by translate or rotate commands or by connecting to the selected grob and dragging/rotating them with the mouse in the graphics window, for example:

```
read grob "icos"
my_icos = g_icos*0.2
display my_icos g_icos
translate add my_icos {5. 15. 0.}
```

You may also create a new or edit an existing graphics objects (see provided \*.gro files to learn about format).

# 3.10.5. Flexible peptide docking

Let us consider a problem of flexible peptide docking to a receptor. The peptide may be completely free or constrained (e.g. have a helical fragment in the middle), and the receptor will be assumed to have a fixed conformation. We will be going through a series of interactive or non-interactive steps. Feel free to put the commands below into several scripts and run them sequentially.

We will follow a particular example through all the steps. In this example we will be docking a constrained 11-residue alpha-helical peptide to a Bcl-XL molecule. It is a good idea to create a separate directory for the project, say, projectX, to keep all the files in one place.

#### Preparing the receptor binding surface for peptide docking.

First, we need to get the receptor model as an ICM object. It may come from several sources. The most common one is a pdb-structure converted with the convertObject macro, or the build model command which builds a model by homology, followed by regularization and refinement with the refineModel macro.

We will create two objects:

a\_1. the original pdb object with receptor with some other peptide a 2. converted receptor (without the peptide)

Example:

```
% cd projectX
% icmgl -G
nice "1g5j" # an open form of the receptor
delete a_w* # delete waters but keep bound peptide
copy a_ "rec"
delete a_rec.2 # keep only the main chain
undisplay window
convertObject a_rec. yes yes yes no # also optimizes hydrogens
#
display box Box( a_1.2 4. ) # display a box around the peptide
```

Now you may rotate the scene and adjust the box size by pulling a box corner with the leftMouseClick. The box needs to cover the receptor surface you need to include in the simulation, but needs to be minimized to reduce the computational burden and memory use.

Sometimes the orientation of the box with respect to the receptor surface is not favorable. In that case, use connect a\_\*. and rotate the objects with the respect to the origin ( use axisLength = 15. and display origin to to see the axis in addition to seeing the box ). If you change the absolute position of the receptor model, you may want to save the new orientation of the pdb object (e.g. write pdb a\_1. "template")

#### The result

Now we have two objects: (i) a pdb object in which we deleted the unnecessary molecules, but kept the peptide for comparison and convenience in the box definition; and (ii) an ICM–object with a receptor for which we will be calculating the grid maps.

#### Create grid potential maps

Now calculate five grid potentials with the make map command:

```
make map potential "gc,gh,ge,gs,gb" Box() 0.5
```

This command uses *all* molecules in the *current object* as a source for the map calculations. The Box() function will return the box you see on your screen. Alternatively you can just provide the 6 numbers

defining that box.

0.5 is the grid size. Even though the ICM grid potential is extremely fast compared to other programs, for large boxes you may need to increase the grid cell size. We do not advise to go beyond 0.7A, however.

The result of this step is five grid maps around the docking surface of interest.

#### Building the peptide

The peptide of interest can be build with the following simple command:

```
buildpep "ECLKRIGDELD" # peptide sequence from Bax
rename a_ "pep"
set a_/* "HHHHHHHHHH"
superimpose a_1.2/310:320/ca a_/1:11/ca align
```

#### Saving the results

Let us now save both the objects and the maps for future use.

write m\_gc m\_gh m\_ge m\_gs m\_gb # gc.map .. files are created write object a\_\*. "all" # write 3 objects in one multi-object file

#### Running a simple simulation from one starting conformation

The actual simulation does not need to be run interactively. An ICM script file \_dockpep is described below. Run the script by

% \_dockpep >! fl.out # the output file

The file contains the following commands:

```
#!/usr/local/bin/icmng
read libraries
read object "all" # contains a_
set object a_pep. # the peptide is the current object
fix v_/3:10/phi,psi,omg
                           # fix some backbone angles
mncallsMC
           = 10000 + Integer(0.008*nvar*nvar*nvar*nvar)
mncalls
            = 170 + nvar * 3
print mncallsMC mncalls
 # make sure that these two numbers are not insane
 # or set smaller mncallsMC and mncalls (e.g. 10000 and 100) as a test
temperature = 600
                   # optimal temperature for the simulation
tolGrad
          = 0.001 # exit minimization when gradient is < 0.001
mnhighEnergy = 15 + nvar/2
set vrestraints a_/* # load preferred backbone and side-chain angle zones
#
vicinity = 2.5
compare a_pep. static # use sRmsd of the peptide
set terms "vw,14,hb,el,to,en,gc,gh,ge,gb,gs"
minimizeMethod = "conjugate"
#
```

```
# maps
read map "gc,gh,ge,gb,gs"
GRID.margin = Distance( a_pep./1/ca a_pep./11/ca )
#
# run montecarlo from a single start
# the reverse option will make more reasonables moves
#
# impose tethers ?
montecarlo reverse movie
# run montecarlo from multiple starts and merge the stacks
for i=1,Nof(<starting conditions>)
montecarlo reverse movie append
endfor
```

This simulation created several files:

fl.cnf a stack file with mnconf (or less) best non-redundant conformations

fl.mov a trajectory file with all accepted conformations (if you used the movie option)

f1.ou the file with the text output of the script

The output file f1.ou contains information about every accepted conformation of the simulation, e.g.

$\overline{\text{DY}} \overline{\text{Visi}}$	300 300		-							1.29 97810 0.57 97911
DY Visi							-32.42			0.06 97999 0.08 98048
	300	J	asp brmc	Guun	ullin	ann	-32.42	-32.43	49	0.08 98048

#### Analyzing the results of a single simulation

#### Energy profile: Is the simulation long enough

The energies can be plotted to analyze the progression of the global energy optimization.

plotEnergy "f1.ou" 50.
plotBestEnergy "f1.ou" 50. "display append"

The first macro plots energies of each conformation generated and accepted in the course of the ICM stochastic global energy optimization. Due to the nature of the procedure, the energy may go up and down. The second macro plots (and appends the plot to the previously generated def.eps file) only the lowest energy achieved by a given iteration. This plot shows if and when a better conformation is found. Naturally, the energy can not be improved after the global minimum is found.

However, having a long flat plateau does not guarantee that the global minimum is found since a new drop of energy may come after a very long simulation. The time required for finding the global minimum in general depends on the number of variables, number of atoms (the cost of a single energy calculation) and complexity of the potential energy surface.

The best empirical way to make sure that the minimum is found is to compare the lowest energies found from completely different starting conformations (read about multiple start simulations below).

We use a window of 50 kcals/mole from the lowest energy. It helps to avoid the dependence of the scale on the initial energy which may be quite high.

To analyze the improvements only use plotBestEnergy.

#### Graphical analysis of the results

You can generate all stack conformations as independent ICM objects with the mkStackConf macro, e.g.:

read object "all" # three objects, the peptide is the current object
mkStackConf 1 10 # make ICM objects from 10 best energy structures

The objects will be called a\_pep1. , a\_pep2., etc. Now these objects can be displayed simultaneously or separately and further analyzed, e.g.

```
display a_pep*.
color a_pep?*. Energy( stack ) # in case you extracted all confs
```

You can loop through the conformations interactively either with the original stack, e.g.

```
# create some view you like
for i=1,10 # or Nof(conf)
    load conf i
    pause  # rotate and zoom, hit Return
endfor
```

, or use the objects you created with the mkStackConf macro.

#### Running a multiple start simulation

You can figure it out yourself :-).

#### convergence analysis

The key question we would like to ask is if at least two independent simulations from random starting conformations identified a nearly identical conformation with a similar energy as the lowest energy conformation. A low-tech example with just two simulations f1 and f2:

```
read object "all"
load conf 0 "f1" # 0 is the lowest energy conformaiton.
load conf 0 "f2"
```

You can also plot the best energy (see above) and compare the plots, as well as perform graphical analysis to see if the best conformations are similar.

#### Analyzing the results of a multiple start simulation

#### merging stacks

Use the read stack append command to merge all stack conformations together. Now you can redefine the compare command and the vicinity parameter depending on how you want to further

compare and filter out the accumulated conformations. The compress command performs the compression.

# 3.11. How-to: Docking and Virtual Ligand Screening

by Max Totrov and Ruben Abagyan

## 3.11.1. Docking and virtual ligand screening. Overview.

This section concerns with predictions of interactions of drugs or small biological substrates (less than about 600–700 dalton) to pockets of larger, more rigid, receptors (typically, protein molecules, DNA or RNA). There are five major steps in docking and screening.

#### Where to dock. Building Receptor and Pocket Model

The **goal** here is to have an adequate three–dimensional model of the receptor pocket you are planning to dock ligands to. And the **pitfalls** are that your model is not accurate overall, or does not reflect the induced fit, or alternative conformations of the receptor binding pocket are missed.

#### Receptor from PDB

If you have only a single entry with your receptor, convert the protein with convertObject yes yes no no, after deleting water molecules and irrelevant chains (e.g. delete a\_!1), or use menus as in the ligand docking section.

However, if you have a choice between several templates, take the following into account:

- X-ray structure is preferable to an NMR structure
- high resolution X-ray structure (less than 2.1A) is much better than, say 2.5A.
- watch out for **high–B–factor** regions and avoid them; sometimes crystallographers deposit fantasy coordinates with high–B–factors. Use:

or Color/B-factor from the Gui-menu.

- place polar hydrogens and choose correct form of histidine (convertObject yes yes no no takes care of that)
- a **bound** conformation of the receptor is preferable, however if you use an apo-model, an NMR structure or a model by homology, the side-chains in a pocket may be incorrect. Frequently they **stick out** and prevent a ligand from binding. Those stubborn side-chains can be 'tamed', (i) manually; (ii) by a side chain simulation with elevated surfaceTension; or (iii) by an explicit flexible docking calculation with a known ligand.

Receptor from homology modeling

A model by homology can be built with the build model command (menu Homology/Build\_Model) followed macro refineModel.

Identifying pockets

3.11. How-to: Docking and Virtual Ligand Screening

If a binding pocket is not known in advance, use icmPocketFinder or icmCavityFinder (for closed pockets) macros. icmPocketFinder can also be accessed from menu Docking/Receptor Setup, submenu Identify\_Binding\_Sites

#### What to dock. Ligand, ligands, a database or a library.

Usually a good start is to try to dock the known ligand(s) to the receptor model. You may also want to dock a library of compounds in order to identify lead candidates. In this case the main **pitfall** is that the library is too restricted, molecules are not chemically feasible or not drug–like.

#### Ligand from PDB

Then to dock a ligand from pdb, go through the procedure described in the ligand docking section.

#### *Ligand(s) from a mol/mol2– file, or SMILES strings.*

The main prerequisite is that the formal charges and the bond types are correct. If they are not correct, you need to process each molecule manually as described in the ligand docking section. From a command line you may use the build smiles or convert2Dto3D macro.

#### Flexible docking considerations.

After the receptor maps are built, you will start a docking simulation. The goal of the flexible docking calculation is prediction of correct binding geometry for each binder. ICM stochastic global optimization algorithm attempts to find the global minimum of the energy function which includes five grid potentials describing interaction of the flexible ligand with the receptor and internal conformational energy of the ligand. During this process a stack of alternative low energy conformations is saved ( one of the choices in the Docking menu ). Some facts about ICM docking:

- an average docking time is 1 3 minutes per ligand per processor
- ICM docking is probably the most accurate predictive tool of the binding geometry today.
- the time per ligand was chosen to be the smallest possible to allow screening of very large data sets. To increase the time spent per ligand, change the Docking\_thoroughness parameter from the Docking.Small Set Docking Batch menu to 3. or 5., or supply this parameter to the rundock script directly.

**Pitfalls.** Inaccurate receptor model, or incorrectly converted ligands, or insufficient optimization effort may lead to incorrect predictions.

#### Scoring

The goal of scoring in virtual ligand screening is to ensure **maximal separation between binders and non-binders**, and **not** to rank a small number of binders according to their binding energies. The **vls** module allows you to access a good scoring function.

## 3.11.2. How-to: Ligand docking simulations.

#### Introduction and pre-requisites

ICM ligand docking procedure performs docking of the fully flexible small-molecule ligand to a known receptor 3D structure. Before setting up the docking project, ICM object of the receptor has to be created. In most cases, x-ray structure of the receptor is initially in the PDB format. Thus, it has to be converted to the ICM format. This process involves addition of the hydrogen atoms, assignment of atom types and charges from the residue templates (icm.res) and imposition of internal coordinates tree (icm-tree) on the original pdb coordinates. The easiest way to convert pdb structure into icm object is through GUI as follows:

- 1. load pdb file into icm (menu File/Read Molecule/PDB)
- 2. convert loaded structure into icm object (menu MolMechanics/ICM-convert/Protein).

It is recommended that "optimize hydrogens" option is selected. To accelerate the procedure, disable the 3D graphics window (side menu Clear/No Graphics) When the procedure finishes, converted object is the 'current' object in icm. You can check the results by displaying the converted structure.

#### Docking project setup

Start project setup by defining the project name (menu Docking/Set project name). Avoid spaces and leading digits in the name. All files related to the docking project will be stored under names, which start from the project name. Most customized parameters will be saved in the table file under the project name as well:

etc..

The next step is to set up the receptor (menu Docking/Receptor setup). Select the receptor molecules, in most cases a\_\* will do – all molecules in the current object will be included. Define binding site residues, either manually e.g. a\_/123,144,152 for selection by residue numbers, or graphically using lasso tool (don't forget to set selection level to residue). This selection is used solely to define boundaries of the docking search and the size of the grids and doesn't have to be complete, selecting some 4 residues delimiting the binding site is sufficient. Receptor setup dialog also lets you run binding site identification routine to quickly locate putative binding sites on your receptor.

After the receptor setup is complete, the program normally displays the receptor with the selected binding site residues highlighted in yellow xstick representation. Ligand setup offers a number of ways (submenu docking/ligand setup) to define the ligand, depending on the source of the ligand structure(s).

#### Converting a chemical from pdb.

The Protein data bank, due to unprecedented ignorance, for the last 15 years has not been storing any information about covalent bond types and formal charges of the chemical compounds interacting with proteins! This oversight makes it *impossible* to *automatically* convert those molecules to anything sensible and requires your manual interactive assignment of bond types and formal charges for each compound in a pdb–entry. Therefore, if you apply the convert command to a pdb–entry with ligands, the ligands will just become some crippled incomplete molecules which can not be further conformationally optimized.

Follow these steps to convert a chemical properly from a pdb form to an a correct icm object.

- display the molecule, set wireStyle=2 (or via top-left gui-menu), and selection type to GRAPHICS.selectionMode=1 (the first item of the gui-selection-mode menu)
- invoke MolMechanics.Structure.SetBondType menu item
- graphically select groups of atoms (e.g. a ring) and set appropriate bond type
- invoke the next menu item, MolMechanics.Structure.SetFormalCharge and set formal charges
- proceed to the MolMechanics.ICM-Convert.Chemical menu (see below)

#### Setting up a ligand or a set of ligands

Let's now consider the situation when icm object of the ligand loaded. ICM object of the ligand can also be prepared, for instance, by reading structure from SD file (menu File/Read Molecule/Mol/SDF) and converting it to ICM (menu MolMechanics/ICM-Convert/Chemical).

Once the icm object of the ligand is ready, proceed to docking ligand setup (menu Docking/Ligand Setup/From Loaded ICM object). The ligand setup procedure will first display the grid box, allowing you to adjust the box dimensions, and then the 'probe' which defines the initial positioning of the ligand's center of mass and long/short axis. The probe can be moved/rotated. While its positioning has only minor influence on the results as long as it remains inside the binding site, it may help the procedure to find the correct docked orientation more reliably and/or in shorter time.

Ligand setup procedure can be ran repeatedly to change the ligand within the same docking project. Also box size and probe position can be changed later (menu Review/Adjust ligand/box). At this point, the project is ready for the calculation of maps (menu docking/Make receptor maps). The calculations generally take several minutes to prepare the maps. While the dialog allows changing the grid step, we do not recommend altering the default value of 0.5 which was found optimal for a large number of test cases. With the map calculations completed, everything is ready to start the actual docking simulation. A larger set of ligands in a mol file can be considered as a **database** and indexed with the ICM indexing tool (menu Docking/Tools/Index Mol,Mol2 Database) for fast access. Ligand structures from mol/mol2 file can be converted to ICM on the fly and do not require manual preparations necessary in the case of PDB structures.

#### **Running docking simulations**

Use menu docking/Small Set Docking Batch to start docking of one or few ligands in the background. You can also view the process interactively (menu docking/Interactive Docking) although it is much slower due to the time spent on drawing the molecules. The results of the batch docking job are saved in the

```
PROJECTNAME_answers*.ob #icm-object file with best solutions for each ligand
PROJECTNAME_*.cnf # icm conformational stack files with multiple docked conf.
PROJECTNAME_*.ou # output file were various messages are stored.
```

Multiple conformations accumulated during the docking of the ligand can be visualized and browsed in ICM (menu Docking/Browse Stack Conformations). Use menu Docking/Display/Preferences to change default graphic representation of ligand/receptor.

#### Rundock UNIX shell script

Docking batch jobs can also be started from the UNIX command line using rundock shell script. The syntax is as follows:

```
rundock <options> <project name>
 options: -f <ligand entry from>" # if using multiple ligand input file
                                 # range of ligands to dock can be selected
      -t <ligand entry to>"
                                 # change the length of MC docking, default is 1.
      -l <thoroughness>
      -n <scanName>
                                 # change the run name in the output files
      -a
                         # force docking and saving of all compounds
      -5
                         # save stack conformations
                         # evaluate score for all stack conformations (slow)
      -S
                        # redirect output to <project name>_from-to.ou
      -0
      -c <output file> # continue interrupted job with <output file>
```

Example:

rundock -f 3 -t 5 -l 3. -s

will thoroughly (3 times longer simulation than default) dock ligands 3 to 5 and save conformational stacks for each one of them.

#### **Template constrained docking**

It is possible to use a template object to tether part of the ligand structure to a preferred position during the docking simulation. Prepare an object file containing the group of atoms to be tethered to. Edit the .tab docking setup file, setting the s\_templateObj field to the name of the template object file (it is 'none' by default). The variable 1\_superByName controls the way correspondence is established between the ligand and template atoms. If it is 'no', chemical substructure search is performed and tethers imposed according to the substructure match. If 1\_superByName is 'yes', simple matching according to atom names is performed.

## 3.11.3. How-to: Virtual Ligand Screening

#### Introduction

Virtual Ligand Screening (VLS) in ICM is performed via docking of each ligand in the database to the receptor structure, with subsequent evaluation of the docked conformation with a binding–score function.

Best-scoring ligands are then stored in the multiple icm-object file. The set-up of the VLS is largely identical to the set-up of the docking simulation (see How-to: Ligand Docking Simulations). In most cases the ligand input file will be an SDF or MOL2 file. These files need to be indexed by ICM before they can be used in VLS runs. The index is used to allow fast access to an arbitrary molecular record in a large file. Use menu Docking/Tools/Index Mol/Mol2 File/Database to generate the index, then set up the SDF/MOL2 file as a ligand source (menu Docking/Ligand Setup/From Database). As in docking, rundock UNIX shell script can be used to start simulations.

#### Score Threshold

An important parameter of the VLS run is the score threshold. Docked conformation for a particular ligand will only be stored by ICM VLS procedure if its binding–score is below the threshold. Edit the project .tab file to adjust this value:

```
#>r DOCK1.r_ScoreThreshold
    -35.
```

The choice of the threshold can be done in two ways:

- 1. based on the scores calculated by docking known ligands. Generally, a value somewhat above typical score observed for known ligands is a good guess.
- 2. if no ligands are known, a pre-simulation can be run using ~1000 compounds from the target database. Using the resulting statistics for the scores, the threshold should be set to retain ~1% of the ligands.

#### Potential of mean force score

Potential of mean force calculation (pmf) provides an independent score of the strength of ligand-receptor interaction. The pmf-parameters are stored in the icm.pmf file. To enable calculation of the pmf-score, define the PROJECTNAME.r\_mfScoreThreshold threshold parameter to the table:

```
#>r PROJECTNAME.r_mfScoreThreshold
999.
```

#### Other selection criteria

ICM VLS uses a number of criteria to pre-select compounds before docking. Edit the project .tab file to change their defaults:

```
#>i DOCK1.i_maxHdonors
5
#>i DOCK1.i_maxLigSize
500
#>i DOCK1.i_maxN0
10
#>i DOCK1.i_maxTorsion
10
#>i DOCK1.i_minLigSize
```

# Parallelization

If the database size exceeds several thousand compounds, it is desirable to run a number of VLS jobs in parallel to speed up calculations. Use –f and –t options of rundock to start multiple jobs on different parts of the database, e.g.

rundock -f 1 -t 10000 -o rundock -f 10001 -t 20000 -o rundock -f 20001 -t 30000 -o

#### Running VLS jobs in PBS UNIX cluster environment

Jobs on the Linux cluster are run through PBS queuing system. Several scripts are provided to facilitate submission of vls jobs. To submit a single job, use pbs script 'pbsrun', which is a pbs wrapper for rundock

qsub \$ICMHOME/pbsrun -v"JOBARGS=-f 1 -t 1000 -o MYPROJECT"

Note that the rundock arguments go in the quotes after JOBARGS= . The qsub command is a part of PBS.

To submit multiple jobs, there is a simple shell script 'pbsscan' which executes multiple qsub's for database stripes:

\$ICMHOME/pbsscan MYPROJECT 1 6000 1000

-submits 6 jobs, 1 to 1000; 1001 to 2000 ... 5001 to 6000. Currently this script only supports default rundock arguments, copy/edit to change.

The command qstat is a part of PBS and can be used to check the status of the jobs. In addition, \$ICMHOME/scanstat script can be used to monitor the progress of the VLS jobs. It analyses the \*.ou rundock output files.

\$ICMHOME/scanstat \*.ou

To delete the jobs, use PBS command qdel:

qdel 1234 # delets job number 1234

#### Where to find the scores of the compounds

Once the compounds are docked, if VLS option is installed, the procedure evaluates the score and stores it in the 'comment' of the ligand object. When browsing scan answers, the SCORE>... line appears for each object viewed, containing the value of the score and it's component terms. It can also be extracted from the icm object in shell using Namex( $a_1$ .) function, and Field() can be used to get particular component or the total: Field(Namex( $a_1$ .) "Score=" 1). The SCORE lines also appear in the output file and can be

extracted by simple unix grep command grep SCORE \*.ou

The MFScore was recently added, it's calculated if r\_mfScoreThreshold variable is defined in the project .tab file. It can be added manually:

```
#>r PROJECTNAME.r_mfScoreThreshold
999.
```

#### Analysis of the results

The hits found by the screening procedure and stored in \*answers\*.ob files can be visualized in ICM (menu Docking/Browse Scan Solutions). If necessary, they can also be exported as SD file using (menu Docking/Tools/Export scan answers as mol). The score and its components are stored in the resulting SD file as well. Simple analysis of the score distribution can be performed by making a histogram (menu Docking/Tools/Scan results histogram).

# 3.12. Example scripts

## 3.12.1. How to predict 3D structure of a peptide from its sequence

In the following script you are going to search for the lowest energy conformation using the Biased Probability Monte Carlo procedure to generate new conformations and full–atom energy plus solvation electrostatics, surface and entropy contributions. Start 3 or more independent simulations and let them run to convergence. Two features are indicative of convergence: the plot of the best energy achieved should be flat for sufficiently long (store the output in f1.ou and run the following macro:

plotBestEnergy "f1" 100. "append display"

); and the lowest energy conformation in different simulations are close, e.g.:

```
# peptide "pep.se" ; runs: "f1" and "f2"
build "pep"
display
read conf "f1" 0
show stack
read conf "f2" 0
show stack
```

Watching movies f1.mov and f2.mov may also be useful. (See also How to evaluate helicity of a peptide from the BPMC simulation and How to calculate an ensemble average). Now, the script:

	# Example folding script. Use as directed.						
read libraries build "pep16"	# your peptide sequence is in pep16.se file.						
rename a_*. "f2"	<pre># specifies current name. # Several runs (f2,f3, etc.) are recommended</pre>						
nvar = Nof( $v_//*$ )	# number of variables						
nProc=4	# if you are using parallel version.						

```
mncallsMC
            = nvar*50000 # maximal number of energy evaluations
mncalls
            = 170+nvar*3 # maximal n_of minimization calls after
                          # each random change
temperature = 600
                     # optimal temperature for the simulation
tolGrad
            = 0.01
                    # exit minimization when gradient is < 0.01
            = 1.0
                     # the default width of the MC probability distributions
mcBell
mnconf
            = 40
                     # maximal n_of low-energy conformations saved
                     # in the stack (f2.cnf file)
mnvisits
            = 25
                    # if stuck for >= 25 times, push it out
mnreject
            = 10
mnhighEnergy = 30
1_bpmc
            = yes
                    # use biased probability
electroMethod = "MIMEL"
surfaceMethod = "constant tension"
set terms "vw,14,hb,el,to,sf,en"
                     # ECEPP/2 energy + solvation + entropy (see icm.hdt file)
fix v //?vt*
                     # exclude irrelevant virtual variables specifying
                     # absolute molecular position
set vrestraint a_/*
                    # load preferred backbone and side-chain angle zones
                     # for the biased probability MC
randomize v_//!omg 180.0 # create random starting conformation
vicinity = 15.0
compare v_//phi,psi
                    # use these variables to compare structure
montecarlo movie
                     # run it and record a movie.
                     # watch the movie later by:
                     # read movie "f2"; display ribbon
                     # display movie "f2" 4. 8.
                     # analyze the best conf. in the stack by:
                     # build "pep16"; read stack; show stack all
                     # load conf 1
```

```
quit
```

# 3.12.2. How to perform local flexible docking of two protein molecules

using the grid potentials

This is a so called "local docking procedure" which docks all orientations of the protein ligand to a certain orientation of the protein receptor. The "global docking procedure" is somewhat different.

You may follow the menu items in Docking.Protein-protein or run the docking scripts directly. To illustrate the principal commands and functions we will also consider a series of shell commands to perform a docking procedure. We will use the following steps from the shell to dock the proteins chymotrypsin (5cha) and APPI (1aap). The real structure of the complex is known (1ca0), which can help us to test the validity of the method. This procedure has been recently tested in a dataset of 24 known protein-protein complexes (Fernández-Recio, Totrov, Abagyan, 2002)

The procedure includes the following steps:

- 1. Creating two ICM objects for both proteins with the convertObject macro
- 2. Specify project parameters in a special table
- 3. Orient molecules, choose the docking box and make potentials.
- 4. Dock the protein ligand into the potentials.
- 5. Refine the solutions.

# 3.12.3. How to perform an explicit flexible docking of two simplified protein molecules

This procedure is relatively old and was used previously to explicitly dock two proteins starting from simplified objects. The best solutions are refined in all-atom representation. Currently we prefer docking into grid (see above).

- 1. Create ICM-objects of the two proteins you want to dock.
- 2. Use macro makeSimpleDockObj to create two simplified objects.
- 3. Combine two simplified objects into one and prepare it for docking simulation using \_makeComplex script. During the execution you will be prompted for orientation of the first molecule, which should face the second one with the expected epitope.
- 4. Run the docking simulation using \_dock2mol script. To insure the completeness of the search, run 3–4 simulations in parallel and compare the resulting stacks, the top 5–7 conformations should be the same except for 1–2. Combine the stacks using "read stack append" command with subsequent filtering by

```
vicinity = 4.
compare static a_2//ca
compress stack
```

- 5. Prepare .var files with optimized surface sidechain conformations for individual proteins by running \_surfSideChainOpt script.
- 6. Run \_makeFullAtom script to create full atom models from the simplified conformations accumulated in the stack.
- 7. Run \_refineComplex script on each of the full atom models
- 8. Complex with the best energy after the optimization is (hopefully) the answer.

## 3.12.4. How to build a model by homology

Have an alignment and a pdb file with the template handy, say "sx.ali" "x.brk". If you have a *homology* module key you can use the build model command and refine the model with the refineModel macro. The build model command builds a complete model and searches for matching loops in all pdb files. You can run the build model command from the GUI interface (menu Homology) alignSS is a good macro to make a sequence-structure alignment. It incorporates solvent accessibility and secondary structure into the alignment procedure. Alternatively, allow the build model command to perform the alignment on the fly.

In the absence of the Homology module, use the following macros/scripts:

- homodel macro: fast interactive model building.
- \_homFast for fast model building (substitute nonidentical side-chains, assign the most likely rotamer).
- \_hom for more rigorous model building for one polypeptide chain: side-chains are optimally placed loops are automatically recognized and simulated.
- \_homMult the same as the above script, but for a multichain protein molecule, e.g. an immunoglobulin molecule. Requires a set of separate files for each alignment.

# 4. References

# 4.1. General literature references

- Abagyan, R.A., Frishman, D., and Argos, P. (1994). Recognition of distantly related proteins through energy calculations. *Proteins* **19**, 132–140.
- Abola, E.E., Bernstein, F.C., Bryant, S.H., Koetzle, T.F., and Weng, J. (1987). Protein Data Bank. In: *Crystallographic databases – Information content, software systems, scientific application* eds. F. H. Allen, G. Bergerhoff, and R. Sievers, Data Commission of the International Union of Crystallography, Bonn/Cambridge/Chester, 107–132.
- Allen, F.H., and Kennard O. (1993) 3D search and research using the Cambridge Structural Database. *Chemical Design Automation News* **8**, pp. 1, 31–37.
- Bernstein, F.C., Koetzle, T.F., Williams, G.J.B., Meyer, Jr., E.F., Brice, M.D., Rodgers, J.R., Kennard, O., Shimanouchi T., and Tasumi, M. (1994). The Protein Data Bank: A computer–based archival file for macromolecular structures recognition of distantly related proteins through energy calculations. *J. Mol. Biol* **112**, 535–542.
- Connolly, M.L. (1983). Analytical molecular surface calculation. J. Appl. Cryst. 16, 548–558.
- Crippen, G.M. and Havel, T.F. (1988). Distance geometry and molecular conformation. Research Study Press, Ltd. (Wiley), New York.
- Eisenberg, D. and McLachlan, A.D. (1986). Solvation energy in protein folding and binding. *Nature* **316**, 199–203.
- Florea, L., Hartzell, G., Zhang, Z., Rubin, G.M., Miller, W., (1998) A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Res* **8**, 967–974.
- Frishman, D. and Argos, P. Incorporation of non-local interactions in protein secondary structure prediction form the amino acid sequence. *Protein Eng.* **9**, 133–142.
- Gonnet, G.H., Cohen, M.A., and Benner, S.A. (1992) Exhaustive matching of the entire protein sequence database. *Science* **256**, 1433–1445.
- Hahn, T. (ed.) (1993) International Tables for Crystallography, Vol. A, D. Reidel, Dordrecht.
- Halgren, T.A. (1995) Merck Molecular Force Field. I.-V. J. Comp. Chem. 17, 490-641.
- Halgren, T.A. (1999) MMFF VI. MMFF94s option for energy minimization studies. J. Comp. Chem. 20, 720–729.
- Henikoff, S. and Henikoff, J.G. (1992). Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA* **89**, 10915–10919.

- Higgins, D.G., Bleasby, A.J., and Fuchs, R. (1992). CLUSTAL V: improved software for multiple sequence alignment. *CABIOS* **8**, 189–191.
- Hutchinson, E.G., and Thornton, J.M. (1994) A revised set of potentials for beta-turn formation in proteins. *Protein Sci.* **3**, 2207–2215.
- Kabsch, W. and Sander, C. (1983). Dictionary of protein secondary structure: pattern recognition of hydrogen bonded and geometrical features. *Biopolymers* 22, 2577–2637.
- Kleywegt, G.J., and Jones, T.A. (1996). Phi/psi-chology: Ramachandran revisited. *Structure* **4**, 1395–1400.
- Kyte, J. and Doolittle, R.F. (1982). J. Mol. Biol. 157 105-132.
- McLachlan, A.D. (1979). Gene duplications in the structural evolution of chymotrypsin. J. Mol. Biol. 128, 49–79.
- Momany, F.A., McGuire, R.F., Burgess, A.W., and Scheraga, H.A. (1975). Energy parameters in polypeptides. VII. Geometric parameters, partial atomic charges, nonbonded interactions, hydrogen bond interactions, and intrinsic torsional potentials for the naturally occurring amino acids. *J. Phys. Chem.* **79**, 2361–2381.
- Needleman, S.B. and Wunsch, C.D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* **48**, 443–453.
- Nemethy, G., Pottle, M.S., and Scheraga, H.A. (1983). Energy parameters in polypeptides. 9. Updating of geometric parameters, nonbonded interactions and hydrogen bond interactions for the naturally occurring amino acids. *J. Phys. Chem.* **87**, 1883–1887.
- Nemethy, G., Gibson, K.D., Palmer, K.A., Yoon, C.N., Paterlini, G., Zagari, A., Rumsey, S., and Scheraga, H.A. (1992). Energy Parameters in Polypeptides. 10. Improved geometric parameters and nonbonded interactions for use in the ECEPP/3 algorithm, with application to proline–containing peptides. *J. Phys. Chem.* **96**, 6472–6484.
- Pearson, W.R., and Lipman, D.J. (1988). Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA* **85**, 2444–2448.
- Saitou, N. and Nei, M. (1987). The neighbor–joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.* **4**, 406–425.
- Shrake, A. and Rupley, J.A. (1973). Environment and exposure to solvent of protein atoms. Lysozyme and insulin. *J. Mol. Biol.* **79**, 351–371.
- Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). Improved sensitivity of profile searches through the use of sequence weights and gap excision. *CABIOS* **10**, 19–30.
- Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). *Clustal W*: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position–specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 22, 4673–4680.

- Veal, J.M. and Wilson W.D. (1991). Modeling of nucleic acid complexes with cationic ligands: A specialized molecular mechanics force field and its application. *J. Biomol. Struct. Dyn.* **8**, 1119–1145.
- Weiner, S.J., Kollman, P.A., Nguyen, D.T., and Case, D.A. (1986). An all atom force field for simulation of proteins and nucleic acids. *J. Comput. Chem.* **7**, 230–252.
- Weininger, D. (1988) SMILES 1. Introduction and Encoding Rules J. Chem. Inf. Comput. Sci., 28, 31.
- Wesson, L. and Eisenberg, D. (1992). Atomic solvation parameters applied to molecular dynamics of proteins in solution. *Protein Sci.* 1, 227–235.
- Wilbur, W.J. and Lipman, D.J. (1984). The context dependent comparison of biological sequences. *SIAM J. Appl. Math.* **44**, 557–567.
- Zhang, X, Mesirov, J.P., and Waltz, D.L. Hybrid system for protein secondary structure prediction. *J. Mol. Biol.* **225**, 1049–1063.

# 4.2. The main description of the ICM method

- Abagyan, R.A. and Totrov, M.M. (1994). Biased probability Monte Carlo conformational searches and electrostatic calculations for peptides and proteins. *J. Mol. Biol.* **235**, 983–1002.
- Abagyan, R.A., Totrov, M.M., and Kuznetsov, D.N. (1994). ICM a new method for protein modeling and design. Applications to docking and structure prediction from the distorted native conformation. *J.Comp.Chem.* **15**, 488–506.

# 4.3. ICM algorithms

- Abagyan, R.A. and Maiorov, V.N. (1988). A simple quantitative representation of polypeptide chain folds: comparison of protein tertiary structures. J. Biomol. Struct. Dyn. 5, 1267–1279.
- Abagyan, R.A. and Mazur, A.K. (1989). New methodology for computer-aided modeling of biomolecular structure and dynamics. 2. Local deformations and cycles. *J. Biomol. Struct. Dyn.* 6, 833–845.
- Abagyan, R.A. and Argos, P. (1992). Optimal protocol and trajectory visualization for conformational searches of peptides and proteins. *J. Mol. Biol.* **225**, 519–532.
- Borchert, T.V., Abagyan, R.A., Kishan, K.V.R., Zeelen, J.Ph., Wierenga, R.K. (1994). The crystal structure of an engineered monomeric triosephosphate isomerase, monoTIM: the correct modeling of an eight–residue loop. *Structure* **1**, 205–213.
- Borchert, T.V., Abagyan, R.A., Jaenicke, R., and Wierenga, R.K. (1994). Design, creation, and characterization of a stable, monomeric triosephosphate isomerase. *Proc. Natl. Acad. Sci USA* **91**, 1515–1518.

- Eisenmenger, F., Argos, P., and Abagyan, R.A. (1993). A method to configure protein side-chains from the main-chain trace in homology modeling. *J. Mol. Biol.* 231, 849–860.
- Mazur, A.K. and Abagyan, R.A. (1989). New methodology for computer-aided modeling of biomolecular structure and dynamics. 1. Non-cyclic structures. *J. Biomol. Struct. Dyn.* **6**, 815–832.
- Totrov, M.M. and Abagyan, R.A. (1994). Efficient parallelization of the energy, surface and derivative calculations for internal coordinate mechanics. *J. Comp. Chem* **15**, 1105–1112.
- Totrov, M.M. and Abagyan, R.A. (1994). Detailed *ab initio* prediction of lysozyme–antibody complex with 1.6 A accuracy. *Nature Struct. Biol.* **1**, 259–263.
- Totrov, M.M., and Abagyan, R.A. (1996). The contour–buildup algorithm to calculate the analytical molecular surface. *J. Struct. Biol.* **115**, 1–6.
- Abagyan, R.A., and Batalov, S. (1997) Do aligned sequences share the same fold? *J. Mol. Biol.*, **273**, 1, 355–368.
- Abagyan, R. and Totrov, M. (1997) Contact Area Difference (CAD): A robust measure to evaluate accuracy of protein models. J. Mol. Biol. 268, 678–285.
- Batalov, S. and Abagyan, R.A., (1999) Universal gap penalty for accurate global-local alignment of biological sequences *J. Mol. Biol.*, **Molsoft report**.

\* Fernandez–Recio, J., Totrov, M., and Abagyan, R. (2002) Soft Protein–Protein Docking in Internal Coordinates. *Protein Science*, **11**.

# 4.4. ICM applications

Some applications of the ICM program are described in the following publications

- Cardozo, T., Totrov, M., and Abagyan, R. (1995) Homology modeling by the ICM method. *Proteins* **23**, 403–414 .
- Borchert, T.V., Kishan, K.V.R., Zeelen, J.Ph., Schliebs, W., Thanki, N., Abagyan, R.A., Jaenicke, R., and Wierenga, R.K. (1995) Three new crystal structures of point mutation variants of monoTIM: Conformational flexibility of loop-1, loop4 and loop8. *Structure* 3, 669–679.
- Strynadka, N.C.J., Eisenstein, M., Katchalski–Katzir, E., Shoichet, B.K., Kuntz, I.D., Abagyan, R., Totrov, M., Janin, J., Cherfils, J., Zimmerman, F., Olson, A., Duncan, B., Rao, M., Jackson, R., Sternberg, M., and James, M.N.G. (1996) Molecular docking programs successfully predict the binding of a beta–lactamase inhibitory protein to TEM–1 beta–lactamase. *Nature Struct. Biol.* **3**, 233–239.
- Chalikian, T.V., Totrov, M.M., Abagyan, R.A., Breslauer, K.J. (1996) The hydration of globular proteins as derived from volume and compressibility measurements: cross correlating thermodynamic and structural data. *J. Mol. Biol.* **260**, 588–603.

- Thanki, N., Zeelen, J.Ph., Mathieu, M., Jaenicke, R., Abagyan, R.A., Wierenga, R.K., and W.Schliebs (1997) Protein engineering with monomeric triosephosphate isomerase (monoTIM): The modeling and structure verification of a seven residue loop. *Protein Eng.* **10**, 159–167.
- Goodman, A.R., Cardozo, T., Abagyan, R.A., Altmeyer, A., Wisniewski, H.G., and Vilcek, J. (1996) Long pentraxins: an emerging group of proteins with diverse functions. *Cytokine Growth Factor Rev.* **7**, 191–202.
- Maiorov, V.N. and Abagyan, R.A. (1997) A new method for modeling large-scale rearrangements of protein domains. *Proteins* 27, 410–424.
- Yu, J., Abagyan, R., Dong, S., Gilbert, A., Nusenzweig, V., Tomlinson, S. (1997) Mapping of the Active Site of CD59. *J. Expt. Medicine* **185**, 745–754.
- Isakoff SJ, Cardozo T, Andreev J, Li Z, Ferguson KM, Abagyan R, Lemmon MA, Aronheim A, Skolnik EY (1998) Identification and analysis of PH domain–containing targets of phosphatidylinositol 3–kinase using a novel in vivo assay in yeast. *EMBO J*, **17**, 5374–5387.
- Maiorov, V.N. and Abagyan, R.A. (1998) Energy strain in three-dimensional protein structures *Folding and Design*, **3**, 259–269.
- Patel, I.R., Attur M.G., Patel R.N., Stuchin S.A., Abagyan R.A., Abramson S.B., Amin A.R. (1998) TNF–alpha convertase enzyme from human arthritis–affected cartilage: isolation of cDNA by differential display, expression of the active enzyme, and regulation of TNF–alpha. *J. Immunol.*, **160**, 4570–4579.
- Srivastava S, Osten P, Vilim FS, Khatri L, Inman G, States B, Daly C, DeSouza S, Abagyan R, Valtschanoff JG, Weinberg RJ, Ziff EB (1998) Novel anchorage of GluR2/3 to the postsynaptic density by the AMPA receptor–binding protein ABP. *Neuron*, **21**, 581–591.
- Zhou, Y., and Abagyan, R.A. (1998) How and why phosphotyrosine-containing peptides bind to the SH2 and PTB domains. *Folding and Design*, **3**, 513–522.
- Abagyan, R., and Totrov, M. (1999) Ab Initio Folding of Peptides by the Optimal–Bias Monte Carlo Minimization Procedure J. Comp. Phys., **151**, 402–421.
- Zhang, H.F., Yu, J., Chen, S., Morgan, B.P., Abagyan, R. and Tomlinson, S. (1999) Identification of the individual residues that determine human CD59 species selective activity. *J.Biol.Chem.*, **274(16)**, 10969–74.
- Zhou, Y., and Abagyan, R.A. (1999) Efficient stochastic global optimization for protein structure prediction *Rigidity Theory and Application, (Thorpe, M., ed.)*, (in press), .
- Schapira, M., Totrov, M., and Abagyan, R. (1999) Prediction of the binding energy for small molecules, peptides and proteins. *J. Mol. Recognition*, **12**, 177–190.
- Stigler, R.D., Hoffmann, B., Abagyan, R., Schneider–Mergener, J. (1999) Soft docking an L and a D peptide to an anticholera toxin antibody using internal coordinate mechanics. *Structure Fold Des*, **7**, 663–670.

- Tomko RP, Johansson CB, Totrov M, Abagyan R, Frisen J, Philipson L (2000) Expression of the adenovirus receptor and its interaction with the fiber knob *Exp Cell Res*, **255**, 47–55.
- Gates MA, Kim L, Egan ES, Cardozo T, Sirotkin HI, Dougan ST, Lashkari D, Abagyan R, Schier AF, Talbot WS (1999) A genetic linkage map for zebrafish: comparative analysis and localization of genes and expressed sequences. *Genome Res.*, **9**, 334–347.
- Li, D., Desai–Yajnik, V., Lo E., Schapira, M., Abagyan, R., Samuels, H.H. (1999) NRIF3 is a novel coactivator mediating functional specificity of nuclear hormone receptors. *Mol Cell Biol* **19**, 7191–7202.
- Schapira, M., Raaka, B.M., Samuels, H.H., Abagyan, R. (2000) Rational discovery of novel nuclear hormone receptor antagonists. *Proc Natl Acad Sci U S A* **97**, 1008–1013.

# 4.5. Credits

Molsoft, L.L.C. has made every effort to supply trademark and copyright information about mentioned individuals, company names, services and products. All products or services not listed below are the trademarks, registered trademarks, service marks, or registered service marks of their respective owners.

- *CSD* is copyrighted by Cambridge Crystallographic Data Centre.
- *DEC* is a registered trademark of Digital Equipment Corp.
- *GhostScript* (gs) is a trademark of Aladdin Enterprises.
- IRIX is registered by Silicon Graphics, Inc.
- IRIS is a trademark of Silicon Graphics, Inc.
- Java and Java –based marks are trademarks or registered trademarks of Sun Microsystems, Inc in the United States and other countries.
- Netscape is a trademark of Netscape Communication Corp.
- *PostScript* is a registered trademark of Adobe Systems, Inc.
- *Prosite* is copyrighted by Amos Bairoch, Medical Biochemistry Department, University of Geneva, Switzerland
- *SD*-*file* and *mol* -file formats are copyrighted by Molecular Design Limited (MDL Information Systems, Inc.).
- Sybyl and mol2 –file format are copyrighted by Tripos, Inc.
- *TIFF* is a trademark of the Aldus Corp.
- UNIX is a registered trademark of UNIX System Laboratories, Inc.
- Wavefront is a trademark of Wavefront Technologies, Inc.
- Windows 95 and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

# 5. Glossary

# 5.1. A

# add

is used as an option equivalent to append in write command.

# alignment

or *sequence group*. Individual sequences in the group may be just combined and left-justified (no insertions/deletions, e.g. as an output of group command) or actually aligned (or realigned) with either pairwise function Align (*seq1*, *seq2*) or multiple alignment command align *sequence\_group* Pairwise sequence alignment is performed with the ZEGA (Zero End-Gap Alignment) algorithm. You can read, write, list, show, or delete an *alignment* unless it is linked to a 3D object. In the latter case you should delete the object first. Flag 1\_showSstructure allows you to show the secondary structure string which belong to a participating sequence to be displayed. When you show the alignment, the consensus string appears on top. The meaning of consensus characters is explained below and the string can be extracted with the Consensus() function.

A table with relative amino acid numbers for all sequences in the alignment is returned by the Table( $ali_{-}$ ) function.

#### Coloring 3D models by local alignment strength. Space averaging

See: selectSphereRadius and ribbonColorStyle = "reliability"

#### Arithmetics:

- extracting a domain (i.e. a certain position range) from an alignment alignment[i1:i2] or alignment[i]
- extracting an alignment of a group of sequences from a larget alignment Align(aliLarge, *I\_seqNumbers*)
- projected alignment: concatenation of two alignments sharing the same sequence. The shared sequence serves as a ruler for merging the two alignments. The alignments can be of arbitrary size and number of sequences. In the simplest case of three sequences a, b, c and alignments ab and bc, the operation ab//bc will create an alignment of three sequences a b c. The function Align(ab//bc,{1,3}) will extract the, so called, projected alignment of **a** and **c** through **b**. Example:

	ali1	11		ali2	returns		Projected	ali.
а	VYRWA-W		b	FK-WG	KW	а	VYR-WA	-W
b	-FKWGKW		С	AKGWA	PGKW	b	-FK-WGK	W
						С	-AKGWAPGK	W

- projecting a numerical property from a sequence to alignment: Rarray( *R\_property,seq\_,ali\_,r\_gapDefault*).
- projecting a numerical property from alignment to a sequence :  $Rarray( \sim R_ali, ali_from, seq_|i_seqNumber)$

Functions:

function name	description
Align	pairwise alignment from 2 sequence, or subalignment extracted from another alignment, or alignment extracted from tethers in an ICM-molecular object.
Consensus()	the consensus string
<pre>Deletion()</pre>	residue selection of deleted residues
<pre>Distance()</pre>	pairwise distance matrix between sequences.
<pre>Insertion()</pre>	residue selection of inserted residues
<pre>Matrix()</pre>	distance matrix
Name()	string array of names of sequences
Nof()	number of sequences
<pre>Profile()</pre>	a profile derived from the alignment
Res()	residue selection corresponding to the aligned sequence
Score()	sequence identity, similarity or alignment score for a pairwise alignment
String()	a multiline string of the alignment (contains '-')
Table()	a table with relative sequence numbers (0 for gaps)
Examples:	
read alignmen	nt "globins.msf"

```
ist alignment "globins.msi"
list alignments
glob_fragment = globins[10:36]
show glob_fragment
delete alignments
```

# all

an option used in several commands where whole list of items involved should be invoked (e.g. show terms).

# alpha helix

alpha-helical conformation. Average angles (phi=-63.2,psi=-38.5). Referred to as 'a' in variable restraint names: e.g. alpha-zone for histidine is called "ha".

See also commands assign sstructure, set vrestraint and file icm.rst.

## amber

a force field for simulations of proteins and DNA in the Cartesian coordinate space.

See the reference.

# append

used as option mostly in read and write commands, for example:

```
write matrix "fil.mat" append
read stack "mc12" append
```

arrays. ICM-shell allows 3 types of arrays:

- iarray integer array,
- rarray real array,
- sarray string array

These arrays are legal elements of the ICM-shell.

**iarray**–constant looks like this:  $\{2, 4, 67, -4\}$ .

**rarray**-constant looks like this:  $\{2.0, -4., .67, -4.3433\}$ .

```
sarray-constant looks like this: { "a-word", "b-word", "c-word", "..." }.
```

Commas are optional unless you have negative elements in integer or real arrays. Array ICM-shell variables can be created by direct assignment: (e.g.  $a = \{2, 4, 67, -4\}$  or  $b = \{"wow", "oops", "ouch"\}$ ), read (e.g. read iarray "numb.iar"), and written to a file (e.g. write numb "numb"). You can specify a subset of an integer array (e.g. a[2:15]). Besides, there all kinds of operations and functions on the ICM arrays. There are many ways to create an array:

- read an array from a file: read rarray "a.dat"
- create by direct assignment:  $a = \{1, -2, 3, 14\}$
- use arithmetic expressions: a=Sin(Cos(a))\*b
- use functions Iarray, Rarray, or Sarray, e.g.: a=Rarray(15, 2.)
- concatenate two elements, e.g.: a=2//3

#### atom

(Greek): indivisible; the smallest particle of a chemical element that can exist alone or in combination (cool, isn't it?).

#### axis

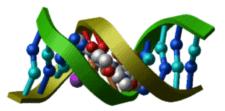
an imaginary straight line passing through a point or a body.

# 5.2. B

## base

Use option base to display cartoon representations of the bases on the DNA/RNA ribbons. In DNA and RNA ribbons, bases can be displayed/colored separately from the ribbon itself (e.g. color ribbon

base  $a_1/*$  white ), the default coloring being A-red, C-cyan, G-blue, T or U-gold.



# ball

a solid sphere representing an atom in the graphics window. Its size is defined by the GRAPHICS.ballRadius ICM-shell variable.

Examples:

See also: xstick

## beta

an extended conformation ("E" in one-character notation). Referred to as 'b' in variable restraint names: e.g. beta-zone for alanine is called "ab".

See also: assign sstructure, set vrestraint and file icm.rst.

# boundary element

See REBEL.See also: electroMethod, delete boundary, show energy", term "el", Potential().

# BPMC

is an acronym for the Biased Probability Monte Carlo method (Abagyan and Totrov, 1994). The method is based on a theorem establishing that if the Monte Carlo procedure is used for global optimization, rather than generation of a Boltzmann ensemble, random moves based on known local probability distributions (e.g. alpha and beta regions for peptide backbone conformations) maximize the optimization efficiency. Practically, the procedure randomly choose a group of coupled angles and changes them according to probabilities defined in the icm.rst file. Do not forget to use the set vrestraint a\_/\* command before the montecarlo command to activate the biasing.

# 5.3. C

## cavity

is the free space inside a molecule. Here we use a limited definition of a cavity as a fully *enclosed* space inside a molecule. See the icmCavityFinder macro which identifies and characterizes cavities.

To generate a cavity an skin is generated and converted into grob. We then split this grob into individual connected entities.

## charge

is an electric charge in electron units. You may redefine charges with the set charge command.

#### coil

is an irregular conformation ("\_" in one-character notation). It is displayed as spaghetti when the ribbon type of representation is used. To assign a residue fragment to coil, do something like this: assign sstructure a\_/13:40 "\_".

## column

indicates a layout format of several arrays (iarrays, rarrays, sarrays) with the same number of elements as side-by-side columns. The column format look like this:

#> a b c
 1. 15 "alpha"
 2. 18 "beta"
 5. 10 "gamma"

It is also useful to show arrays of a table (see also database).

#### current map

usually is the last map loaded by the read map command or created by the make map command. The current map may be changed by the set map command. To find out which one is the current, use list maps.

## current object

usually is the last object loaded by the read object command or created by the build command. The current object may be changed by the set object [*os\_object*] command. To find out which is the current one, use list object or show object [*os\_object*].

You may refer to the current object as a\_

#### current table

usually is the last table loaded by the read table command or created by the group table command. The current table may be changed by the set table command. To find out which one is the current, use list *tables*.

#### command

an instruction one can execute in the ICM-shell interactively or from an ICM script file. Typically a command consists of a verb (like read or delete) and a bunch of arguments. The word order in the argument list is not important, if arguments have different types. For example:

```
display a_//ca,c,n yellow # is as good as ...
display yellow a_//ca,c,n # ... inverse
```

There are exceptions,

• if two or more arguments are of the same type, e.g.

write "This line" "filename"
or superimpose as select1 as select2,

• a complex argument consists of two parts: e.g. delete label *i\_number* or grid["x"] in the plot command. In the latter case if the "x" string is not preceded by grid word, it will not be considered.

#### comp\_matrix

a residue comparison matrix used by the alignment algorithms. The matrix is loaded automatically together with other library files from the icm.cmp file and can be reloaded later (if you need another matrix) using the read comp\_matrix command. The current matrix can be shown. All elements are properly reordered and multiplied by a factor such that the sum of occurrence-weighted diagonal elements (identities) of the matrix is 1.0 and stored in this form. You can increase all the values in the matrix (a useful operation for low sequence identities) with the set comp\_matrix command which has similar effect to reduction of the gapOpen parameter. The set comp\_matrix command also allows to modify weight for specific residue pairs (e.g. set comp\_matrix 4. "CC"). Two matrices are recommended: the blosum50 with penalties 2.0/0.15 and the Gonnet matrix (the default) with penalties 2.4/0.15. A number of matrices are provided. You can use your own data file too.

#### conf

a conformation (actually a set of variables determining the conformation) stored in an ICM stack. You can add a conformation to the conformational stack or load a conformation from the stack.

Example:

```
% icm
buildpep "ala his trp"
montecarlo
show stack
             2
                  3
iconf> 1
                        4
                              5
                                    6
                                            7
ener> -15.1 -14.6 -14.6 -14.2 -13.9 -11.4
                                          -1.7
rmsd> 0.3 39.2 48.0 44.1 27.4 56.6 39.3
naft> 1 0 0 1 1 1 0
naft> 1
nvis> 4
            1
4
                                            1
print Nof(conf)
```

```
7
print Iarray(stack) # returns {4 1 1 4 4 4 1}
t = Table(stack) # more conf parameters
for i=1,Nof(conf)
    load conf i
    center
    pause # hit return
endfor
```

# cpk

abbreviation of "Corey-Pauling-Koltun" who introduced space-filling solid models of atoms. In ICM graphics it means solid representation of van der Waals spheres. Spheres are drawn at van der Waals radii. See also: display

# 5.4. D

## database

a format for storing entries containing several different types of information. The column format (multicolumn representation) is the alternative. The database should have the following format:

FieldName1 Number String FieldName2 Number String FieldName3 Number String FieldName1 Number String FieldName2 Number String FieldName3 Number String ...

Fields can contain integers, reals and multiline strings.

Example:

id jurl iq 24 addr LA addr California id jur2 iq 17 addr LA

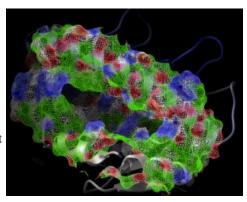
# Depth-cueing, or fog

a visual effect which makes objects disappear in a fog

according to the screen Z-coordinate. Press

Ctrl-D

to invoke the effect and move the back clipping closer with MidMB to make the effect more dramatic. Also more the front clipping plane with Ctrl-MidMB to make the front brighter.



If you use gui adjust the front and back clipping planes using the buttons in the graphics tool bar and vertical movements with the left-mouse-button

Two parameters influence the **fog** effect:

- fogStart determines the relative Z-position at which the linear fog effect starts (the distance between the front and the back planes is 1.)
- color volume *color* command allows to color fog differently from the background (by default they are the same).

## distance

a measure of proximity in three-dimensional or multidimensional space for all kinds of objects, including sequences and real arrays. See function Distance ().

# distance geometry

is often used to calculate a set of Cartesian coordinates of a series of objects given all (or even a subset of) the inter-object distances (Crippen and Havel, 1988). ICM function Disgeo() allows to solve this sort of problems. See also Distance().

# disulfide bond

(or SS-bond) a covalent bond between sulfur atoms of two cystein residues. In the PDB objects loaded by the read pdb command these bonds are automatically imposed on the basis of SSBOND records. Upon conversion of a PDB structure into the ICM-object with the convert command, the disulfide bonds are inherited. **Important:** commands make disulfide bond and delete disulfide bond are valid for ICM-type molecular objects only (and have no effect on, say, PDB structures). Both commands change the covalent structure of the modeled molecular object and expel/add hydrogens. In addition, SS-bonded **cys** residues are renamed to **cyss.** Therefore, to count the number of disulfide bonds you may simply count the **cyss** residues: show Nof(a\_/cyss)/2. Distance restraints imposed to form such a bond are defined in the file icm.cnt.

## drestraint

a distance restraint (sometimes abbreviated as **cn**) which imposes a penalty function on the distance between two atoms of the *same* object ( in contrast, tether imposes a restraint to an atom in *another* object). Distance restraints may have different characteristics and types. A drestraint is a basin with flat bottom between the upper and lower bounds (  $D_u$  and  $D_L$ ) and bi-quadratic walls:

 $\begin{array}{ll} P(d) = 0.25^* (d^2 - {D_u}^2)^2 / {D_u}^2 & \text{if } d > D_u \text{ (upper wall)} \\ P(d) = 0 & \text{if } D_{L < d < D_u u} \text{ (flat bottom)} \\ P(d) = 0.25^* (d^2 - {D_L}^2)^2 / {D_L}^2 & \text{if } d < D_L \text{ (lower wall)} \\ \text{drestraints are mostly used in structure determination using the NMR NOE data.} \end{array}$ 

Drestraints may be read, written, made, setn, shown, and displayed. The parameters of drestraints are written in \*.cn files. See also: icm.cnt file description.

To activate the drestaint penalty term, use the set terms "cn" command.

## drestraint type

a type of distance restraint(s). It describes well depth, lower and upper boundaries and has two forms: *global* and *local*. Local restraints become weaker and vanish as the distance between the corresponding atoms grows (similar to the van der Waals forces), while global restraints become stronger as you deviate further from the required distance range. Local type has an additional characteristic: sharpness. The types are given in \*.cnt files.

The main list of types consists of general user-defined global and local types as well as a number of fixed named types used to impose disulfide bonds or peptide bonds. Drestraint types may be read, written, set, and shown.

See also: \* . cn file description.

# 5.5. E–H

## ecepp

a force field used in the ICM program. The latest version is called ECEPP/3 reported in Nemethy et al. (1992). See also the following references: Momany et al. (1975), Nemethy et al. (1983)

#### fasta

program **FASTA** (Pearson and Lipman, 1988) is used for search sequence databases, evaluate similarity scores and identify sequence similarities on the basis of local sequence similarity. The program is well suited for rapid database searches, because it does not handle insertions/deletions. In ICM, **fasta** also specifies one of the several allowed formats of sequence data storage and representation.

# 5.6. S

#### site

ICM sequences and objects may contain specific information about local sequence features, such as location of binding sites, disulfide bonds etc. These information is stored in the feature table (FT) section of the Swissprot protein sequence entries or after the SITE fields of pdb files. The sites in the feature table may look like this:

$\mathbf{FT}$	ACT_SITE	15	15	ACTIVE SITE HIS
FT	TRANSMEM	309	332	PROBABLE
FT	DOMAIN	333	362	CYTOPLASMIC TAIL.
FT	DISULFID	125	188	BY SIMILARITY.

We use one letter code (the second column) to specify the site type. The first column shows the priority value which is used by the set site command.

Priority Char SWISSPROT def. Description

Thomy	Chai	Swissi Koi uci.	Description
4	А	ACT_SITE	Amino acid(s) involved in the Activity of an enzyme.
2	В	BINDING	<b>B</b> inding site for any chem.group(co–enzyme,prosthetic group)
5	С	CA_BIND	Extent of a Calcium-binding region.
5	D	DNA_BIND	Extent of a <b>D</b> NA-binding region.
4	F	SITE	Any other <b>Feature</b> on the sequence (i.e. SITE records in PDB).
2	G	CARBOHYD	Glycosylation site.
7	Ι	INIT_MET	The sequence is known to start with an initiator methionine.
2	L	LIPID	Covalent binding of a Lipidic moiety
2	Μ	METAL	Binding site for a Metal ion.
5	Ν	NP_BIND	Extent of a Nucleotide phosphate binding region.
6	0	PROPEP	Extent of a pr <b>O</b> peptide.
6	Р	PEPTIDE	Extent of a released active Peptide.
5	R	REPEAT	Extent of an internal sequence Repetition.
6	S	SIGNAL	Extent of a Signal sequence (prepeptide).
5	Т	TRANSMEM	Extent of a Transmembrane region.
1	V	VARIANT	Authors report that sequence Variants exist.
1	Х	CONFLICT	Different papers report differing sequences.
5	Ζ	ZN_FING	Extent of a Zinc finger region.
6	c	CHAIN	Extent of a polypeptide Chain in the mature protein.
5	d	DOMAIN	Extent of a <b>D</b> omain of interest on the sequence.
3	e	THIOLEST	ThiolEster bond.
1	m	MUTAGEN	Site which has been experimentally altered.
2	р	MOD_RES	Post-translational modification of a residue.
3	s	DISULFID	DiSulfide bond.
3	t	THIOETH	Thioether bond.

1	v	VARSPLIC	Sequence Variants produced by alternative splicing.
6	Z	TRANSIT	Transit peptide(mitochondrial,chloroplastic,cyanelle,microbody)
5	~	SIMILAR	Extent of a similarity with another protein sequence.
4	_	NON_CONS	Non consecutive residues.
7	+	NON_TER	The residue at an extremity of seq. is not the terminal res.
4	?	UNSURE	Uncertainties in the sequence

```
The sites can be
```

- read from a swissprot entry with the read sequence swiss command
- set to a sequence or a molecular object with the

```
set site [seq_from [ali_] {seq_|ms_} [only]
```

command

• a new site can be set with the

```
set site s_siteString {seq_|ms_} [only]
```

command (e.g. set site a\_1.1 "FT SITE 15 15 important residue").

• and delete with the

delete site {seq\_|ms\_} i\_siteNumber

command (e.g. delete site a\_mol1 1).

 $\bullet$  To show sequence sites use the show sequence swiss command, and in objects: show site {seq\_|ms\_}

command.

- Sites assigned to molecular objects can be selected (and thereby visualized) with the a\_/ F SiteString selection
- Sites will be written to an object and restored upon reading under the OBJECT.site or OBJECT.auto preference.

The ICM-shell variable 1\_showSites toggles the appearance of the site information in the show sequence command.

Example:

```
read pdb "lhla" # this object Ca atoms of 2 molecules
make bond a_//ca # link them into a chain
rinx SWISS #rinx is an alias to read index "...."
read seq swiss SWISS.ID=="lA02_HUMAN"
read seq swiss SWISS.ID=="B2MG_HUMAN"
read seq swiss SWISS.ID=="lA68_HUMAN"
set site a_1 lA02_HUMAN
```

```
set site a_1 1A68_HUMAN append
set site a_2 B2MG_HUMAN
show site B2MG_HUMAN
cool a_
ds cpk magenta a_/FV # display variants
ds cpk yellow a_/Fs # display disulfides
```

## grob

an abbreviation for a general GRaphics OBject, which contains dots, and/or lines and/or solid surfaces; it can be a geometrical body, a contoured electron density, 3D plot, an arrow, etc. If the graphics object contains triangles, it can be represented by solid surfaces. The order of points in the triangles defines the direction of the normals which in turn defines which of the two sides are lit. Grob–file format is straightforward and editable.

To merge two or several grobs, use the write grob append command.

Example:

```
read grob "icos"  # several example graphics objects
read grob "cube"  # are read in ...
read grob "oblate"
read grob "prolate"
display g_cube red  # ... and displayed
display g_oblate green
display g_prolate magenta
center
```

# hbond- hydrogen bonds

are calculated according to ECEPP/3 potential. The can be displayed. Related commands are show hbond, display hbond, color, and undisplay.

# svariablei, or ICM-shell variable

a named object stored in the program memory of one of the following types: integer (i), real (r), string (s), logical (l), preference (p), iarray (I), rarray (R), sarray (S), matrix (M), sequence (seq), profile (prf). alignments (ali), maps (m), graphics objects (grob) (g). They can be created by direct assignment to a constant (e.g. a={1 4 3 8}, to a function (e.g. a=Iarray(4)) or read from a disk file (e.g. read iarray "a") Most of ICM-shell variables can also be written to a disk file, and shown. They can take part in the arithmetic and logical expressions. For some of the variable types, subsets are defined (e.g. a[2:4]).

# integer

numbers may exist in the ICM-shell as a named variable or a constant (e.g. 123,2,-45). There are several dozen predefined integer variables. Integers may be mentioned in arithmetic expressions, commands and functions.

Examples:

## label

usually a string displayed in the ICM graphics window. Types of labels:

- atom label # toggled by LeftMB clicking
- residue label # toggled by LeftMB double clicking
- variable label
- free string label # drag it with the Middle mouse button.

To display the free text label, type:

display "Below lies a black abyss"

To delete it delete label *i\_labelNumber* e.g. delete label 2

To show:

show labels

# logical

may exist in ICM-shell as a named variable or a constant (only two possibilities: yes and no) You can use exclamation mark for negation (!) and two operations: *and* ( & ) and *or* (|) There is a number of predefined logical variables. Logicals can be used in arithmetic expressions, most frequently in if (*logical*) ... expressions.

Examples:

#### macro

a group of ICM commands in a separate named function. See description macro in the command section.

#### map

a real function defined on a three-dimensional grid. Usually it is an electron density map or grid potential. See also: icm.map This ICM-shell object contains a descriptor (or header) with the following information:

• cell type (space group number) and parameters {a, b, c, alpha, beta, gamma};

- lattice and sublattice specifications (sizes and offsets for columns, rows and sections);
- characteristics of the density values: the mean value, standard deviation, the minimum and the maximum values.
- correspondence between X,Y,Z and sections, rows and columns

The map itself contains a stream of real density values for each node of the sublattice.

Maps can be read, calculated from structure factors, and created as a result of map arithmetics. Maps of 5 types of grid potentials can also be calculated with the make map potential command. The last map loaded or created becomes **the current map**. The current map is a convenient default for commands requiring map as an argument.

The following arithmetic operations between maps of compatible sizes are allowed: *map+map*, *map+i*, *map+r*, *i+map*, *r+map* map-map, map-*i*, map-*r*, *i-map*, *r-map* map\**r*, map\**i*, *i\*map*, *r\*map*, map\*map map/r, map/i.

#### Map functions:

Box(map)	returns R_6box defining the map boundary
Bracket	same as Trim
Cell(map)	returns R_6 crystallographic cell parameters of the map
Map( map I_3_or_6 [ simple ] )	a submap
Min(map)	minimal value
Max(map)	maximal value
Nof(map)	total number of grid points
Rarray(map)	returns all values from 3D-grid points as a linear real array
Smooth( map [ expand] )	space-average the map values
Symgroup(map)	string with the symmetry group name
Trim( map,vMin vMax )	trim by values outside the range
Trim(map,R_6box)	set values <b>outside</b> the box to zero
Simple arithmetic operations are a	allowed with the maps (map1 and map2 must have the same dimensions):

• plus (map1 + map2, map + r),

- minus (map1 map2, map r),
- multiply (map1 \* map2, map\*r, r\*map ),
- divide (map1 / map2, map/r),

One can also use expressions, e.g.

 $m = Smooth(m_ge)*2. - 1. + m_gc \# does not make much sense$ 

#### matrix

a set of real numbers organized in rows and columns. The ICM-shell allows arbitrary size matrices [n,m], access to its elements (M[i,j]), rows (M[i]), columns (M[1:i,j]) or any submatrix (M[i1:i2,j1:j2]). Basic matrix operations such as

- plus (M1 + M2),
- minus (M1 M2),
- multiply (M1 \* M2),
- concatenate rows ( M1 // M2 ),
- equal ( M1 == M2 ),
- not equal ( M1 != M2),
- Transpose(M), and
- inverse ( Power(M,-1) )

allow powerful matrix arithmetics. You can **create** a new matrix in the ICM-shell by reading (read matrix "a"), assignment (M\_new=Transpose(M\_old)) or function Matrix (e.g. M=Matrix(4,8)). Matrix-related functions are the following:

- determinant of square matrix ( Det(M))
- principal components or "distance geometry" (Disgeo (M)) function, i.e. if a given square matrix M[1:n,1:n] contains distances between n points find coordinates in (n-1) dimensional space and sort the space dimensions according to their contribution to the variation. If distances are 3-dimensional Euclidean distances, the first three coordinates will give you x,y,z.
- Eigen (M) function returns a matrix of eigenvectors; eigenvalues are stored in R\_out
- Distance( *alignment*) returns matrix of pairwise distances between sequences in an alignment.
- Max, Min, Mean and Sum functions return a row (actually a real array) with maximal, minimal, mean, or total values in each column, respectively
- Nof(M) and Length(M) return *n* and *m*, respectively for matrix *M*[1:*n*,1:*m*].
- Power(M, *i\_exponent*) calculates different integer powers of a matrix, including matrix **inverse** ( inmat=Power(M, -1)),
- Random(d1,d2,n,m) creates a matrix and fills it with random numbers,
- Rmsd(M) returns root-mean-square deviation,
- Trace(M) returns the trace of a square matrix,
- Xyz( *as\_select*) returns a matrix of xyz coordinates of selected atoms.
- Distance( matrix) returns a matrix of pairwise distances between the row-vectors of the matrix.

#### Matrix assignments

Examples:

By simple arithmetic operations with matrices you can

- solve a system of linear equations ( x=Power(A,-1)\*B ),
- find best set of parameters x[1:m] which fits your model A[1:n,1:m] (n > m) to data vector B[1:n]. Minimum of (A\*x-B) is found by 3 steps:

```
M1=Transpose(A)*A
M2=Power(M1,-1)
x =(M2*Transpose(A))*B
```

## MIMEL

an abbreviation of Modified IMage ELectrostatics algorithm (Abagyan and Totrov, 1994) developed for fast evaluation of both internal Coulomb and electrostatic polarization free energy for large molecules. This term has no analytical derivatives and has no effect on local energy minimization. It can be a part of the energy function in global optimization such as montecarlo or ssearch. Three components of MIMEL can be shown using the show energy command. They are:

- Coulomb interactions of explicit atomic charges (note that it is divided by the dielConst ICM-shell parameter)
- "Self energy" (or interaction of explicit charges with their own images)
- "Cross energy" (or interaction of explicit charges with other charges' images)

The last two components together represent the electrostatic polarization energy which is returned in the  $r_{out}$  variable. REBEL gives a more accurate evaluation of the electrostatic solvation. For small molecules use mimelDepth = 0.3. For proteins the error in the solvation energy evaluation (returned in the  $r_{out}$  variable) is estimated as 15 - 20%.

#### mmff .

This word refers to the Merck molecular force field described in a series of 1994 and 1999 publications by Thomas Halgren. ICM can assign MMFF atom types using local chemical environment, formal charges and 3D topology. ICM also allows to calculate the mmff94 energy and minimize it both in the cartesian space with free covalent geometry and in the internal coordinate space with fixed covalent geometry or user-defined geometrical constraints.

See also:

- ffMethod defines which force field to use
- read library mmff sets the parameters set type mmff identifies the atom types from the covalent geometry, formal charges, and bond types

• minimize

#### mol

This word refers to the MDL Information Systems, Inc. SD–file format for small molecules (see trademarks). ICM can read and write molecules in this format. They may look like this:

```
name
jscorina 12209406473DS
LongName
 76
                       0.0104 C
   -0.0187
           1.5258
                                   0 0 0 0
                                               0
                       0.0020 C
                                   0 0 0
   0.0021
           -0.0041
                                            0
                                               0
                      -0.0024 S 0 0 0
   1.6831
             2.1537
                                            0
                                               0
                      0.0129 C 0 0 0 0
           -0.5336
   -1.4333
                                               0
                      -1.7665 C 0 0 0 0
0.0045 C 0 0 0 0
-2.5386 C 0 0 0 0
             1.9811
                                               0
    2.0692
   -1.4126 -2.0635
                                               0
  1.4620 3.1542
2 1 1 0 0 0
                                               0
  3 1 1 0 0 0
```

```
4
    2 1 0 0
               0
 5
    3
       1 0 0
               0
 6
    4
      1 0 0
               0
 7
    5 1 0 0
               0
> <NSC>
19
> <CAS RN>
638-46-0
$$$$
```

## mol2

This word refers to the Tripos file format for small molecules (see trademarks). ICM can read and write molecules in this format. The default extension for this type of file is .ml2. They may look like this:

```
@<TRIPOS> MOLECULE
a1
    3
         2
SMALL
USER_CHARGES
@<TRIPOS>ATOM
                                                1 hoh
                                                          0.3280
    1 ho1
            -2.0000
                      0.0000
                              -1.0000
                                      H
                                                1 hoh
    2 o
             -2.4944
                      0.0000 -1.8229 O
                                                          -0.6550
    3 ho2
             -3.4149
                      0.0000
                              -1.5503 н
                                                1 hoh
                                                          0.3280
@<TRIPOS>BOND
                2
                     1
    1
         1
    2
          2
                3
                     1
```

#### more

an internal ICM-viewer, a little brother of the UNIX browser with the same name. Displays ICM output one screenful at a time. Control:

- spacebar: next page
- Return: scroll by one line
- /string: find string
- n find next
- q: quit

## movie

a series of molecular conformations representing a Monte Carlo trajectory and saved in an ICM-formatted .mov binary file can be simply displayed or used for animated.

The icm .mov files are *not* quicktime movies, or series of images. Instead, they contain a compressed series of geometrical parameters determining object geometry for each accepted montecarlo iteration.

The *frames* of the trajectory/movie file can be separately analyzed and further filtered with an ICM script. For example, one can generate a shorter movie by retaining only the frames with lower energies.

See also: display movie, load frame

#### mute

an option in a number of commands (e.g. find pattern, find prosite, show tether, show energy, show area, show volume, etc). It is usually used in scripts when one wants to suppresses unnecessary output. In macro declaration, this option suppresses prompting for missing macro arguments.

## only

frequent option in commands which means disregard or delete the previous status. Without only commands usually add or append to the current settings.

Examples:

#### pattern

a sequence consensus pattern like this, "[AG]?[!P]W", or this "C?G?\ $\{2,3\}$ C". A pattern can be extracted from an alignment and searched against a sequence database. See also:

- find pattern find a pattern in a single sequence,
- find database pattern efficient parallel pattern search in a BLAST-formatted sequence databank.
- Pattern( *s\_consensus*) create a regular pattern expression from a consensus,
- Pattern( *alignment*) create a regular pattern expression from an alignment,
- regular expressions and pattern matching
- prosite a collection of sequence patterns

#### png

graphics image format. Stands for Portable Network Graphics and was designed to replace the GIF format and, to some extent, the much more complex TIFF format. While GIF allows for only 256 palette colors, PNG can handle a variety of color schemes like TIF (1,3,8, 24, etc. bit colors). Furthermore, PNG is free, while GIF is subject to licensing fees. PNG also supports alpha–channel. Since 1998 most browsers correctly display PNG images. See also: rgb, tif.

**Pattern matching and regular expressions.** Use the following metacharacters to construct regular expressions (try guess what string is used in the examples!)

- \* matches any string including an empty string (e.g. "\*see\*")
- ? matches any single character (e.g. "???ee M")

• [string] matches any one of the enclosed characters. Two characters separated by dash represent a range of characters.

Examples: [A-Z], [a-Z], [a-z], [0-9] (e.g. "[A-Z] see [A-Z]"

- [!string] negation. matches any but the enclosed characters (e.g. "I see [!K]")
- single-character multiplication: *character*\(m,n) (e.g. "I?\{3,6\}M" repeat any character, ?, from 3 to 6 times)

The example string was "I see M". Regular expressions may be used in selections ( $a_*.//c?,n,c$ ), and in list, group, delete commands. Note that for the latter three commands the pattern must be quoted.

## pdb or Protein Data Bank

a repository of macromolecular structures solved by crystallography or NMR (occasional theoretical models are frowned upon). It used to be at the Brookhaven National Laboratory, Now it is shared between UCSD and Rutgers University. The old citations: Bernstein et al., 1977; Abola et al., 1987). The new citations can be found at http://www.rcsb.org/pdb/. On November 20th, 2001 it contained 16596 entries.

An example ATOM record:

ATOM 52 N HIS D 18 53.555 24.250 49.573 1.00 32.59

## peptide bond

a covalent bond between C=O and N-H groups, which is imposed in ICM-objects as an extra set of distance restraints. These groups may belong to the terminal groups as to the amino acid side chains. **Important**: commands make peptide bond and delete peptide bond are valid for ICM-type molecular objects only (and have no effect on, say, PDB structures). Both commands change the covalent structure of the modeled molecular object and expel/add hydrogens. Distance restraints imposed to form such a bond are defined in icm.cnt file.

## profile

a table of residue preferences for each residue type at each position on a protein fold or a sequence. The preferences may be derived from a multiple sequence alignment of from a 3D structure. Profile also contains gap opening and gap extension values for each sequence position. Profile provides a good way of representing a consensus sequence pattern of a protein family. One can search a new sequence against a library of profiles, or search a profile against a data base of protein sequences (see Abagyan, Frishman, and Argos, 1994). One can add two profiles (prf1 + prf2), multiply them (prf1 \* prf2), concatenate two profiles (prf1/prf2), and extract a part of a profile (prf[15:67]). Profile can be read from a .prf file and calculated from an alignment with the Profile() function. See also: Sequence() Consensus() Align().

## prosite

a dictionary of protein sites and patterns, (Copyright by Amos Bairoch, Medical Biochemistry Department, University of Geneva, Switzerland). ICM converts prosite patterns to standard string patterns containing regular expressions, like "C?\{4,5\}CCS??G?CG????[FYW]C".

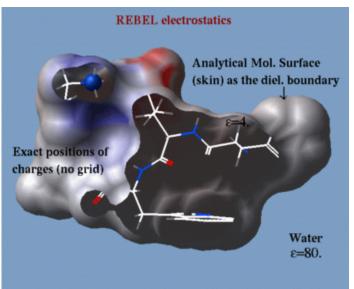
The old releases of prosite can be found at ftp://ftp.expasy.org/databases/swiss-prot/sw\_old\_releases/

See also:

- read prosite,
- •s\_prositeDat,
- find prosite find all prosite patterns in a single sequence,
- find profile find all prosite profiles in a single sequence

## REBEL

a method to solve the Poisson equation for a molecule. REBEL is a new powerful implementation of the boundary element method with analytical molecular surface as dielectric boundary. This method is fast (takes seconds for a protein) and accurate. REBEL stands for Rapid Exact-Boundary ELectrostatics. The energy calculated by this method consists of the Coulomb energy and the solvation energy which is returned in the  $r_out$ system variable.



Related parameters and steps:

- electroMethod = "boundary element";
- dielConst (the default is usually OK);
- dielConstExtern (the default is usually OK);
- set charge *as\_r\_Charge* (modify charges if you like);

- make boundary (if you want to make several evaluations of energy or Potential() with the same boundary. The "boundary" parameters depend only on conformation and do NOT depend on charges. You can redefine charges afterwards and get a corrects energy evaluation);
- delete boundary (if you do not need it);
- show energy (make sure the "el" term is on);
- Potential (*as\_targets as\_charges*) (if the boundary exists, returns potentials from charges at the target atoms);
- color grob potential (create graphics object, say, with make grob skin actually it can be any grob and color it by the REBEL potential);

The polarization charges can be returned by the Rarray(as\_) function after the equation is solved, e.g.:

```
electroMethod = 4
show energy "el"
```

```
Rarray( a_//* )
```

#### real

number may exist in the ICM-shell as a named variable or a constant (e.g. 12.3, 2.0, -4.501). There are a number of predefined real variables. Reals may be mentioned in arithmetic expressions, commands and functions.

Examples:

a = -1.2 b = Abs(Sin( 2.3 \* a - 3.0 / a))

## regularization

procedure for fitting a protein model with the ideal covalent geometry of residues (as represented in the icm.res residue library) to the atom positions of a target PDB structure (usually provided by X-ray crystallography or NMR). Regularization is required because the experimentally determined PDB-structures often lack hydrogen atoms and positional errors may result in the unrealistic van der Waals energy even if these structures were energetically refined (since the refinement of the crystallographic structures typically ignores hydrogen atoms and employs different force fields). The following steps are required to create the regularized and energy refined ICM-model of an experimental structure:

- an extended all-atom model of a particular protein is generated with regular geometry characteristics (see the build command and the IcmSequence function);
- the non-hydrogen atoms in the model are assigned to the equivalent atoms in the model (see set tether);
- the regularized structure is built starting from the N-terminus by adding atoms one-by-one (see minimize tether);
- methyl groups are rotated to reduce van-der-Waals clashes;
- combined geometry and energy function is optimized;
- polar hydrogen positions are adjusted;

• optionally the model may be additionally minimized, now without tethers to observe a "stability" of the model in the local energy minimum.

See macro regul.

# residue

a chemical building block or complete chemical compound, usually an amino-acid residue. The ICM hierarchy: atom -> residue -> molecule -> object. Individual small molecules may contain only one residue. Residues are described in the icm.res file. You may create your own residues with the write library command. Residues can be selected with the ICM-selection expression (e.g. a\_/ala, a\_/15, a\_/15:20, a\_/"RDGE" etc.), labeled with the display residue label *rs\_* command, by double clicking with the right mouse button, via a pop-up menu, or from the GUI menu.

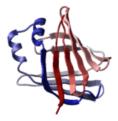
## rgb

red-green-blue. It is of interest, that the combination of these three can produce any other color. In addition, this is the name of the SGI image format used in the ICM commands write image and display movie. ICM also generates the fourth channel on top of the RGB information. This fourth number is called alpha-channel and generates the opacity index for each pixel of the image. This information is interpreted by a number of applications, i.e. the IRIX showcase and dmconvert (the SGI moviemaker). See also tif, targa, postscript.

## ribbon

a graphical representation of a polypeptide chain backbone by a smooth solid ribbon. DNA and RNA can be also displayed in a ribbon style.

There are three types of elements of the ribbon display depending on the secondary structure assigned to a given residue.



Residues marked as alpha-helices ('H') will be shown by a flat ribbon, those marked as beta-sheets ('E') will be flat ribbon with an arrow-head, and the rest will be shown by a cylindrical "worm". The ICM-shell parameter GRAPHICS.wormRadius defines its radius. Default ribbon colors are defined in the icm.clr file. Note that minor secondary structure elements like 3/10 helix ('G'), Pi-helix ('I') and 'Y' are colored by the "alphaRibbon" color, 'L','P' and 'B' (isolated beta-residue) residues are colored by the "betaRibbon" color. DNA and RNA ribbons are colored according to the base type: A-red, C-cyan, G-blue, T or U - gold. Preference ribbonStyle allows to display a simplified segment representation of the secondary structure elements instead of (or together with) the ribbon.

The DNA/RNA ribbons consists of two parts the backbone ribbon and the bases shown with the sticks and balls. To selectively display and undisplay the bases, you can do the following:

Example:

## script

(or ICM script) means a collection of ICM commands stored in a file which can be called from ICM-shell.

Example:

call \_demo\_fold # find demo\_fold file and start the script

#### sequence

an ICM-shell object containing an amino-acid or DNA sequence. The ICM-shell is tuned to work with very large sets sets of millions of genomic sequences at once. To work with the sets larger than 2 Gigabytes in size use the 64-bit binary executable (it is standard on Cray and Dec, unavailable on Windows and optional on SGI). One can read a sequence from a sequence file in different formats, create it with the Sequence() function, make sequence command, or by assignment (e.g., aseq = bseq [2:18], new sequence aseq is a 2:18 fragment of sequence bseq). A valid amino-acid sequence contains an uppercase string of one-characters amino-acid names. Please distinguish this ICM-shell object from the "sequence" in the ICM-sequence file which contains detailed 3 (or 4)-character notations of residues from the icm residue library. One can concatenate two sequences (seq1 // seq2 ) and extract a part of it (seq[15:67]). Sequence object may contain the secondary structure string (e.g. EEE\_\_\_HHH\_\_) of the same length as the sequence. It is automatically created by the make sequence command and the Sequence() function or can be directly set with the set sstructure command. If logical 1\_showSstructure is set to yes, the secondary structure string will be shown in alignments.

Examples:

```
aseq=Sequence("ASSAARTYIP")
read sequences "aa.seq"
aseq[3:4]="WW"
read object "crn"
crn seg = Sequence(a /*)
```

#### **Resetting sequence type**

ICM is trying to guess sequence type. To set sequence type explicitly, use the set type [protein|nucleotide] command. E.g.

```
a=Sequence("AAAATAAAA")
set type a protein # or if you change your mind
set type a nucleotide
```

Properties of a sequence can be projected to an alignment in which the sequence participates with the Rarray( $R_property, seq\_ali\_, r\_gapDefault$ ) function. The opposite action, i.e. projecting from alignment to a particular sequence can be achieved with another form of the Rarray function: Rarray( $R\_ali\_ali\_from, seq\_i\_seqNumber$ )

#### segment

an element of the simplified representation of a protein topology in terms of its secondary structure elements (Abagyan and Maiorov, 1988). One element (referred to as a segment) is a vector of the best axis of the element. Loop segments are represented by a straight line between the end of the previous segment and the beginning of the next one. This representation can be used for a fold search through a library of precalculated segment descriptions of the protein topologies (foldbank.seg). See also ribbonStyle.

# (ICM)-shell

user-friendly, high-level command interpreter combined with a collection of tools allowing you to interact conveniently with the kernel of the ICM software.

## skin

a solid graphical representation of the molecular surface, also referred to as the Connolly surface. It is a smooth envelope touching the van der Waals surface of atoms as the solvent probe of the waterRadius size rolls over the molecule. "Skin" is important for analysis of recognition, electrostatics, energetics, ligand binding and protein cavities. The surface is calculated with a new fast analytical **contour-buildup** algorithm (Totrov and Abagyan, 1996) and can be generated as a general graphics object with the make grob skin command. 'Skin' consists of three types of elements: convex spherical elements, concave spherical elements, and torus-shaped elements. ICM allows the calculation of the volume confined by the 'skin' and its surface area. In a general case skin is defined by **two** atom-selections:

- 1. atoms the skin is calculated for
- 2. atoms surrounding the atoms from the previous selection

One can calculate/display only a patch within a context of the rest (as\_part a\_\*), or skin around one molecule as the rest does not exist (as\_part as\_part):

```
read object "complex"
display a_//ca,c,n
pocket = a_1//!h* Sphere(a_2//!h*)
display skin pocket a_1//!h* # 5A sphere around the second subunit
set plane 2  # or F2 : to avoid deletion of the previous patch
display skin a_2//!h* a_2//!h* green # ignore everything but the second molecule
```

Colored molecular surface can be saved as:

- bitmap image (tif, targa, gif, postscript bitmap) (write image "file")
- vectorized postscript containing triangles, not pixels (write postscript "file")
- the skin can be converted into a uniform color grob (make grob skin)
- the skin can be colored by potential and ((color grob potential try also: show dsRebel)

• Warning: make grob image does not generate correct normals because of a feature in OpenGL, however, this command works fine for molecular representations such as cpk, ribbon, xstick, etc.

ICM can also generate smooth gaussian surfaces with the following commands:

make map potential Box( a\_ 3. ) # build gaussian map make grob m\_atoms solid exact 0.5 # contour it display g\_atoms # display the envelope grob

#### smiles

Simplified Molecular Input Line Entry Specification. The acronym introduced by David Weininger to represent chemical valence model by a string (e.g. CC=O). It can also be used as an exchange format for chemical data. The algorithm was published in 1988 and is described in detail at the WWW site of Daylight Chemical Information Systems, Inc.

See also the Smiles function and the build smiles command.

#### sln

Sybyl line notation, a string representation of molecular structure similar to Smiles. The sln string is returned by the  $String(as\_sln)$  function.

#### stack

a set of conformations of a particular object. The stack can be just a place to store (with the store conf command) a number of complete descriptions of different conformations regardless of the way they have been created. The maximal number of stack conformations is determined by the mnconf parameter. The stack conformations can be created manually in the course of interactive procedure, or created automatically as a result of a montecarlo run. The energies of stack conformations can be shown with the show stack [all] command. The stack can be saved into a .cnf file, and you can also read stack. Stack in Biased Probability Monte Carlo procedure represents best energy representatives of different conformational families (see Abagyan and Argos, 1992). Measure of difference (or distance) is defined by the compare command and vicinity parameter. Stack can influence the search via the following variables: mnvisits, mnhighEnergy, mnreject, visitsAction, highEnergyAction and rejectAction.

#### See also:

- conf (has good examples),
- Table(stack) stack conformation parameters
- Iarray(stack) (the number of visits to each stack conformation).
- Nof (conf)
- load conf i

# stick

graphical representation of a covalent bond as a solid cylinder. Its radius is defined by the GRAPHICS.stickRadius ICM-shell variable.

# string

may exist in the ICM-shell as a named variable or a constant (e.g. "lcrn", "A b\n c"). There is a number of predefined string variables in the ICM-shell. You can concatenate strings ("aaa" +''bbb" or "aaa" //''bbb" -> "aaabbb"), sum a string and a number ("aaa" +4.5 -> "aaa4.5"), compare them (if (s\_pdbDir == "/data/pdb/", or if (s1 > s2)). Strings may be used in arithmetic expressions, commands and functions.

Examples:

```
s = "lcrn"
sl = sl + ".brk"
if (s != "2ins") print "wrong protein"
```

# structure factor (factor)

a named ICM-shell table containing information about reflections. A structure factor table header may contain maximal absolute values of h k and l.

#>I igd.HKL
 31 36 37

It will be calculated on the fly if absent and is important for Fourier transformation. You may also have any number of additional members in the header section for your convenience. For example, real values for the minimal and maximal resolution, etc.

The "column" part of a table contains mandatory integer arrays of h,k and l. Some of the other arrays with fixed names may be necessary for specific operations. They are:

- fo: real array of observed amplitudes (used by the "xr" term)
- $\bullet~fc:{\tt real}~{\tt array}~of$  calculated amplitudes. They are added and updated automatically by the "xr" term calculations.
- ac and bc : real array of Real and Imaginary components of calculated structure factors. ac and bc may be read from a file, calculated in the ICM-session, and/or added and updated automatically by the "xr" term calculations. These two arrays are used as the input arrays for the make map factor command.
- w:real array of weights of individual reflections which are used if defined in the "xr" term calculations. Note, that multiplicity will be automatically taken into account, do not multiply your weights by it to avoid double counting.
- free : integer array of 0 and non-zeros to mark reflections for R-free calculations. Reflections marked with non-zeros will not be used in the "xr" term calculations. They

will be used instead by the Rfree(*T\_factor*) function.

One can add any number of additional arrays to the factor-table. Of course, the table can be read, written, sorted, shown, etc. You may also use powerful table arithmetics and expressions to generate new columns and specify subsets.

Examples:

See also: How to manipulate with structure factors

The command word "factor" serves to read/write the XPLOR formatted structure-factor-files.

#### surface area

in the ICM-shell means a solvent-accessible surface (center of water-sphere). **Important**: Do not confuse this surface with the molecular or Connolly surface which is referred to as skin . (see also Acc function, Area function, display skin, display surface, show area surface, show area skin, show volume surface "sf" term).

**Important:** There are two ways to calculate the surface area: via the show area surface or the show energy "sf" commands. In both cases individual atomic accessibilities are calculated and assigned to individual atoms. These accessibilities can be shown with the show as\_command, or can be accessed with the Area(*as\_*) function. However, the two commands use different atomic radii:

- show area surface
  - uses van der Waals radii as defined in the icm.vwt file
  - calculates areas for all atoms including hydrogens
- show energy "sf"
  - uses special radii designed for calculations of the solvation energy. The radii are defined in the icm.hdt file;
  - employs a united atom model, in which hydrogens are ignored and radii increased accordingly;
  - ♦ calculates areas only for non-hydrogen atoms, ignores hydrogens.

Examples:

```
show a_//*  # look at the accessibilities
show Area(a_//n*)  # extract atomic accessibilities for all nitrogens
#
show energy "sf"  # only heavy atom accessibilities used in energy calc.
show a_//*  # look at these new accessibilities
show Area(a_//n*)  # "energy" accessibilities for nitrogens
```

# 5.7. T

#### table

an ICM object which unite several other ICM-objects. It consists of two parts:

- a header which is a set of any ICM-objects and
- a spreadsheet composed of iarrays, rarrays, or sarrays of the same length.

Tables can be read, written, and shown. Tables can also be created by

group table *obj1 obj2* ...

command and returned by some functions such as:

- Energy(stack) returns a table with energy values for the stack conformations
- Table(s\_out) interprets an output of an HTML form.
- Find performs search in all entries and returns the matching entries

#### Pairwise table expressions:

- !(Table selection) negation
- T.I\_? i\_(? is one of: ==, !=, <=, >=, <, >)
- T.R\_? r\_(? is one of: ==, !=, <=, >=, <, >)
- T.S\_? s\_(? is one of: ==, !=, ~, !~, i.e. exact of fuzzy comparisons)
- T.S\_? S\_(? is one of: ==,  $\sim$ , equivalent to T.S\_? S\_[1] | T.S\_? S\_[2] | ...)
- T.S\_? S\_(? is one of: !=, !~, a complement to what is returned by the T.S\_ == S\_ or T.S\_ ~ S\_ comparison, respectively, equivalent to T.S\_ ? S\_[1] T.S\_ S\_[2] ...)

The result of the pairwise expression is a subset of the table involved. The pairwise expressions can be further combined with the & (AND) or | (OR) operations. A full expression can be assigned to a new table or used on the fly in a number of commands and functions. String comparison can use patterns (e.g. t.NA=="\*\_MOUSE"). The pattern may contain more sophisticated  $\{n,m\}$  expressions, if the first symbol is '\*' or '^'.

Example:

```
group table t {"a", "b", "c"} "s" {1 2 3} "i" # arrays t.s, t.i show t show t.s == {"c", "a"} # shows the 1st and the 3rd lines show t.s ~ "a*" | t.i < 3 # shows the 1st and the 2nd lines
```

## table subsets:

Table subsets can also be defined explicitly through the three types of index expressions:

• T[i\_element], e.g. t[3]

- T[i\_from:i\_to], e.g. t[3:15]
- T[I\_indexArray], e.g. t[{3,14,18}]

Index arrays are returned by some commands or the Iarray(T\_) function.

Look at this example of operations with tables. We read a database of secondary structures foldbank.db dump arrays into a table, add sequence length to a table, extract entries of interest, sort them and save the result.

```
read database "foldbank.db"
                                      # load information into arrays
LE=Length(SS )
                                      # create iarray with sequence lengths
group table t $s_out LE
                                      # create table t with all info + lengths
                                     # press 'q' otherwise computer will explode
show t
show t.NA == {"lgec.i","5pad*"}
                                     # find these entries
a=t.RZ < 2.2 t.ER < 1. t.LE > 35 # select entries with resolution < 2.5,
                                     # converted with ER < 1. and longer</pre>
                                     # than 35 residues
sort a.LE a.RZ
                                      # resort entries according to
                                     # lengths/resolution
write database a "SUBSET"
```

#### tether

a harmonic restraint pulling an atom in the current object to a static point in space. This point is represented by an atom in another object. Typically, it is used to relate the geometry of an ICM molecular object with that of, say, an X-ray structure whose geometry is considered as a target (see also delete tether, minimize tether, show tether, set tether).

The restraint can also pull an atom to a z-plane (rather than to a point), if you specify tzMethod="z\_only"

Atom specific weights can be imposed with tzMethod="weighted" via bfactors.

Two other types of restraints are drestraint (distance restraints), and vrestraint (multidimensional variable restraints).

#### tif files

Tag(ged) Image File Format, used by default in the ICM commands write image and display movie. See also: rgb, png, targa.

## transformation vector

an elementary space transformation is defined by a rarray where values  $\{a1,a2,...,a12\}$  define 3x3 rotation matrix and translation vector  $\{a4,a8,a12\}$ . The complete augmented affine 4x4 transformation matrix in direct space can be presented as:

a1	a2	a3	a4
a5	a6	a7	a8
a9	a10	a11	a12
0.	0.	0.	1.

The commands and functions related to transformation vector (referred to as  $R_tv$ ):

- transform *ms\_R\_tv* applies transformation to an object;
- Symgroup(*i\_spaceGroupNumber*) returns a chain (R\_[1:12\*n]) of all n transformation vectors composing the specified space group;
- Augment  $(R_tv)$  converts 12-membered transformation vector into the augmented transformation matrix 4x4;
- Vector( $M_4x4$ ) converts a 4 by 4 transformation matrix into 12-membered transformation vector
- superimpose *as\_1 as\_2* ... returns *R\_tv* in R\_out;
- Rmsd( *as\_1 as\_2* [ exact ] ) returns *R\_tv* in R\_out;
- Axis( $R_tv$ ) calculates the rotation axis  $R_3$  of the transformation. Rotation angle is returned in r\_out;
- Rot(*R\_tv*) extracts the 3x3 rotation matrix;
- Trans( $R_tv$ ) extracts the translation 3-vector which is applied after rotation.

# 5.8. U–Z

## unique

an option of the group sequence command used to exclude from the group identical sequences (which may result from, e.g., from a set of PBD structures with the same sequence, but somewhat different conformation).

## virtual atoms and variables

Additional immaterial geometrical points (referred to as "virtual atoms") attached to each molecule for technical reasons, and internal coordinates ("virtual bonds, angles, torsions and phases") associated with them. These points help to have a standard yet flexible treatment of parameters defining absolute position (translation and rotation) of each molecule with respect to the coordinate frame. Each molecule is connected to the origin via two virtual atoms attached to it. This part of the ICM–molecular tree is built in the following way:

- **o2**, **o1**, **o** three points with coordinates  $\{1, 0, 1\}$ ,  $\{0, 0, 1\}$  and  $\{0, 0, 0\}$ , respectively. They are the same for all molecules
- vt1 the first virtual atom of a molecule. It is attached to the origin (o) via virtual bond length bvt1, planar angle avt1 (o1-o-vt1) and a dihedral angle. The dihedral angle is a torsion angle tvt1 (o2-o1-o-vt1) for the first molecule in the tree, but it is a phase angle fvt1 (a difference between dihedrals o2-o1-o-vt1( *current-molecule*) and o2-o1-o-vt1( *lst-molecule*)) for all molecules but the first one.

- vt2 the second virtual atom attached to vt1 via virtual bond length bvt2 (usually fixed), planar angle avt2 (o-vt1-vt2) and a a torsion angle tvt2 (o1-o-vt1-vt2).
- the first real atom of a molecule. Torsion angle leading to it is called tvt3.

The absolute position of the first molecule as a rigid body is defined by six virtual variables:

```
Name StdValue Type Definition

1. tvt1 180. torsion **02-01-0-vt1

2. avt1 90. angle **01-0-vt1

3. bvt1 1. bond **0-vt1

4. tvt2 180. torsion **01-0-vt1-vt2

5. avt2 90. angle **0-vt1-vt2

6. tvt3 180. torsion **0-vt1-vt2-1stAtom
```

The

set a\_1//vt1

command sets the first virtual atom to the center of mass of the corresponding molecule. After that you can control the distance to the origin by vtl. To understand it better, try the following short session (just paste it line by line):

```
build string "se ala\nml a\nse his" # two molecules
display virtual
varLabelStyle = "name"
display variable labels
display virtual atom labels a_//vt*
connect a_1 # move the 1st molecule with the mouse
connect a_2 # move the 2nd molecule with the mouse
set a_1//vt1
set a_2//vt1
set v_1//bvt1 2.
```

## volume

measured in cubic Angstroms. One can calculate the van der Waals volume (see Volume function), the volume confined by solvent-accessible surface (see show volume surface), or by molecular surface referred to as skin (see show volume skin).

#### vrestraint

a multidimensional variable restraint (often abbreviated as **rs**) which restrains one or several geometrical variables (usually torsions) to certain ellipsoidal zones, described by Abagyan, Totrov and Kuznetsov (1994). Variable restraints may have different characteristics and types. The vrestraints are marked either for energy calculations or for description of probability distributions (fields 'rse' and 'rs', respectively). See also: icm.rs and icm.rst files.

## vrestraint type

a type of multidimensional variable restraint. Each type specifies to which variables this type may be assigned, the average values and standard deviations (or well dimensions), and the well depth. The types

are described in \*.rst files. There are two kinds of vrestraint types depending on what they will be used for: energy kind marked with the 'rse' field and the probability type marked with the 'rs' field. The first kind will be used as an "rs" penalty term, while the second kind will be used for the BPMC random step.

#### wire

a default representation of a molecule, fast and simple. Bonds are shown by lines or arrows according to the wireStyle preference. Double bonds in the wireStyle "chemistry" mode are shown according to the wireBondSeparation parameter. Isolated atoms are shown according to the atomSingleStyle preference (usually by a small tetrahedron); line thinkness is controlled by the lineWidth parameter.

Examples:

```
display a_lcrn./n,ca,c # displays a wire model of crambin.
    # (note, display command can also
    # read in the 3D coordinates
    # if double " " used)
```

You would need to mention "wire" explicitly to undisplay it when other types of graphical representation are also present.

Examples:

```
display # wire is the default
display cpk a_/1:5 # adds CPK
undisplay wire # remove wires, leave only CPK
```

## xstick

a combination of ball and stick representations of atoms and bonds.

# ZEGA

a pairwise alignment method based on the Needleman and Wunsch algorithm modified to use zero gap end penalties. This type of alignment was first described by Michael Waterman, who called it the "fit" alignment. The paper of Abagyan and Batalov, 1997 describes the statistics of the structural significance of the alignment score and optimization of the alignment parameters for the best recognition of structurally related proteins. This statistics is used in database search (see the find database command) to evaluate the significance of hits. This pairwise alignment algorithm is used in the Align function, align command, and database searching.

# Index

	54	alignment editor	35,160
14 term	61	alignment extraction	293
3D plots intro	22	alignment options	273
3D smoothing	477	alignment projection	480
=	51	alignment score	457
aba92	479	alignment sequence reordering	293
abb97	479	alignment structural	125,293
abbreviations	25	alignment to sequence transfer	350
abm88	479	alignment to text conversion	480
abm89	479	alignment transitions	51
abo87	477	alignment weighted	293
Abs	291	alignment.gapExtension	74
Acc	292	alignment.gapOpen	74
accessible surface	492	alignment.transitional	56
accFunction	119	alignMethod	94
Acos	293	alignMinCoverage	71
Acosh	293	alignMinMethod	71
add	483	alignOldStatWeight	71
addBfactor	71	all	477
advanced operations	56	all torsions table	436
af	61	al193	477
afa94	477	alpha	483
alias	122	alpha channel	110
align	123	amber	483
Align	293	amidation	448
align 3D heavy	127	and	54
align 3D how to	479	Angle	297
align fragments	124	animation	480
align number	123	append	483
align sequences	123	Area	297
aligned residues	438	arithmetic operations	51
alignment	273	arithmetics	51
alignment block length	70	array appending	51
alignment cleaning	438	array concatentation	51

array derivative	477	binding energy	480
array.subset	37	binding pocket finding	439
as2_out	122	blast files	284
Asin	299	bok95	480
Asinh	299	Boltzmann	438
Ask	300	bond angle bending	410
assign	128	bond stretching	410
assign sstructure	128	bonded atoms	482
assign sstructure segment	70	bor93	479
assignment	51	bor94	479
as_graph	45,105	boundary element	485
as_out	121	Box	438
Atan	300	BPMC	485
Atan2	482	Bracket	438
Atanh	482	break	130
ato94	479	bs	61
Atom	482	build	130
atom	483	build from sequence	130
atom code file	410	build from string	136
atom type	380	build helix	459
atom user field	482	build how to	446
atomLabelStyle	93	build hydrogen	136
atoms.alternative position	240	build loop	134
atoms.selecting	45	build model	131,279
atoms.translate	237	build smiles	134
atomSingleStyle	94	build string	136
Augment	427	buildpep	446
autoSavePeriod	65	Cad	479,480
Axis	477	cad97	479
axis	483	calcBindingEnergy	480
axisLength	72	calcDihedral4atoms	480
baa99	479	calcDihedralAngle	480
background.color	140	calcEnsembleAver	480
ball	485	calcMaps	457
base	485	calcPepHelicity	457
ber77	477	calcProtUnfoldingEnergy	460
beta	485	calcRmsd	460
Bfactor	477	calcSeqContent	460

calculate phases	460	color by charge	477
call	136	color by electrostatic potential	66
call ICM script	136	color by hydrophobicity	430
car95	480	color cursor	140
cavity	486	color file	479
cavity analysis	437	color grob	140
cd	241	color grob by electrostatic potential	140
Ceil	480	color grob by map	140
Cell	480	color grob unique	140
center	137	color grob vertices	140
cha96	480	color label	142
change atom position	237	color map	142
change unix directory	241	color molecule	143
Charge	457	color ribbon	143
charge	486	color volume	143
charge.change	239	column	486
chemical clustering	460	combine transformations	459
chemical formula	460	command	486
chemical keys	195	command line editing	31
chemical matching	172	command line help	183
chemical modification	203	command line options	31
chemical substructure	172	command word list	427
chemical substructure mask	195	compare	68,69,144
chemical superposition	61	compare by atoms	144
clear	137	compare by contact surface	144
clear screen	137	compare by variables	144
Cluster	457	comparison operations	55
cn	61	compress stack	145
cnWeight	73	comp_matrix	241,261,486
coil	486	con83	477
color	137,138	concatenation	51
Color	460	conf	477, 486
color background	140	conf data	477
color background example	429	configuration	479
color by accessibility	430	configure.memory usage	67
color by bfactor	430	conformational generator	201
•		e	

		crystal symmetry	
conformational stack	486	transformation	458
conformational stack file	477	crystallographic occupancy	477
connect	146	crystallographic symmetry intro	17
Connolly surface	492	current	486
Consensus	460	current icm-process number	66
consensus.selecting residues by	43	current map	486
conservation selection	43	current object	486
constants	36	customization	482
continue	147	database	489
contouring density	194	dc	61
convert	147,468	dcMethod	94
convert and reroot	150	dcWeight	73
convert comparison	148	debugging shell scripts	211
convert ICM object to PDB	272	defCell	119
convert object macro	460	define axis	458
convert pdb	447	defSymGroup	66
convert to icm-object	147	delete	151
converting a chemical	149	delete alias	151
converting a pdb-chemical	468	delete atom	152
converting alignment to table	477	delete bond	153
cool	400	delete boundary	154
coordinate frame	72,160	delete conf	154
сору	150	delete disulfide bond	155
copy object	150	delete drestraint	154
Corr	310	delete hydrogen	152
correlation matrix	438	delete label	154
Cos	310	delete molecule	153
Cosh	482	delete object	152
Count	482	delete peptide bond	155
covalent neighbors	482	delete selection	152
cpk	486	delete sequence	154
create a covalent bond	189	delete shell object	151
credits	482	delete site	155
cri88	477	delete sstructure	155
crypt	150	delete stack	156

delete table	156	display tethers	167
delete term	156	display window	167
delete tether	156	Distance	477
Deletion	482	distance	489
density correlation	73, 477	Distance alignment	438
density fitting	94	Distance as_	477
densityCutoff	73	Distance as_ rarray	477
depth cueing	143,489	distance contact-based	479
depth-cueing	74	Distance Dayhoff	438
Det	427	distance geometry	489
dielConst	73	Distance iarray	477
dielConstExtern	73	Distance matrix	477
dielectric constant	73	Distance rarray	477
dihedral angle calculation	436	distance restraint	489
Disgeo	427	distance restraint file	477
display	156	distance restraint term	61
display box	160	distance restraint type file	477
display clash	161	distance restraints	241
display cursor	160	Distance tether	438
display drestraint	162	Distance two alignments	479
display from script	159	distribution and support	12
display gradient	162	distribution comparison	457
display grob	162	disulfide bond	448
display hbond	163	disulfide bond formation	448
display label	163	DNA alignment	124
display map	164	DNA Representation	14
display model	156	dna to protein sequence translation	457
display movie	164	doc94	479
display new	159	docking result viewing	480
display off-screen	159	docking simple models	476
display origin	160	drestraint	261,489
display ribbon	166	drestraint generate from structure	191
display site	166	drestraint global weight	73
display skin	166	drestraint set	241
display string	166	drestraint type	489
display surface	166	drop	73
-			

ds3D	457	en	61
dsCell	482	endfor	168
dsCellBox	482	endif	168
dsCharge	427	endmacro	168
dsChem	427	endwhile	168
dsConsensus	427	energetics	479
dsCustom	427	Energy	446
dsCustomFull	477	energy terms	61
dsDistance	477	energy.electrostatic	61
dsEnergyStrain	438	energy.hydrophobic	61
dsEnergyStrain1	479	energy.side-chain entropy loss	61
dsPropertySkin	477	energy.torsion	61
dsPrositePdb	446	ensemble average	480
dsRebel	66,446	entry atom	150
dsSeqPdbOutput	480	Error	480
dsSkinLabel	480	error ignoring	91
dsSkinPocket	480	errorAction	96
dsStackConf	480	EST-alignment	124
dsVarLabels	480	et vrestraint vs_var	251
dsWorm	457	evolutionary tree intro	21
dsXyz	457	example scripts	480
ecepp	491	Exist	457
edit	168	Existenv	460
Eigen	479	exit	169
eis86	477	exit action	102
eis93	479	Exp	460
el	61	expressions.arithmetics	51
electroMethod	95	expressions.assignment	51
electron density fitting	61	expressions.comparison	55
electron density map generation	71	expressions.logical	54
electrostatic isopotential surfaces	194	Extension	460
electrostatic solvation	479	factor	492
electrostatics intro	18	fast Fourier transform	196
ellipsoid	438	fasta	491
elseif	167	ffMethod	96

Field	320, 482	FTP.createFile	104
File	482	FTP.keepFile	104
file exists	457	FTP.proxy	104
files	408	functions.selecting in objects	50
FILTER	104	fuzzy comparison	55
FILTER.gz	104	gap expansion	438
FILTER.uue	104	gapExtension	74
FILTER.Z	104	gapFunction	119
Find	427	gapOpen	74
find alignment	169	gat99	480
find database	71,170	gb	61
find family of commands	169	gc	61
find pattern	174	ge	61
find pdb	173	genomics clustering	178
find prosite	174	genomics intro	19
find segment	175	geometry optimization	201
findFuncMin	460	Getenv	477
findFuncZero	400	getting started	26
fit to density	94	gh	61
fix	176	glossary	483
flattening 3D molecule	427	glycosylation	448
flo98	477	goc96	480
Floor	477	gon92	477
fog	143,489	goto	177
fogStart	74	Gradient	477
fold search	175,492	graphical box	160
foldbank.db	409	GRAPHICS	105
foldbank.seg	409	graphics	427
folding procedure	480	graphics controls	32, 479, 489
font size	479	graphics exists	457
for	176	graphics intro	13,15
fork	176	graphics learning	427
formatdb	284	graphics.attributes	72
fprintf	177	GRAPHICS.ballRadius	105
fri96	477	GRAPHICS.dnaBallRadius	105
fta02	479	GRAPHICS.dnaRibbonRatio	105
FTP	104	GRAPHICS.dnaRibbonWidth	105

GRAPHICS.dnaRibbonWorm	105	group table	180
GRAPHICS.dnaStickRadius	105	gs	61
graphics.fogStart	74	GUI	109
GRAPHICS.quality	105	gui	181
GRAPHICS.rainbowBarStyle	105	gui exists	457
GRAPHICS.resLabelDrag	105	gui programming	181
GRAPHICS.ribbonRatio	105	GUI.workspaceTabStyle	109
GRAPHICS.ribbonWidth	105	hah93	477
GRAPHICS.ribbonWorm	105	hal95	477
GRAPHICS.selectionLevel	105	hal99	477
GRAPHICS.selectionStyle	105	haze	74
GRAPHICS.stickRadius	105	hb	61,74
graphics.view vector	251	hbCutoff	74
GRAPHICS.wormRadius	105	hbond	492
GRID	108	hbond.show	263
grid potentials	197	hbonds how to	479
GRID.gcghExteriorPenalty	108	heat	102
GRID.margin	108	hei92	477
GRID.maxEl	108	helix content calculation	482
GRID.maxVw	108	help	183
GRID.minEl	108	help commands	183
GROB	109	help functions	183
grob	438	help.getting	482
Grob	438	hidden display	159
grob	492	hig92	477
grob color	438	highEnergyAction	98
grob files	479	Histogram	446
grob inside-out flip	244	history	184
grob normal directions	244	history of ICM	12
grob vertex selection	438	homodel	400
GROB.arrowRadius	109	homology modeling	131,476
GROB.relArrowHead	109	homology modeling intro	16
GROB.relArrowSize	109	hut94	477
grob.translate	274	hydration parameters	479
group	177	hydrogen bonding	61
group replacement	203	hydrogen bonding parameters	479
group sequence	177	hydrogen bonding.cutoff	74
group sequence unique	178	hydrophobicity profile	437

hybrid synthol110110117iarray480icm.htm479larray480icm.iar479larray reverse480icm.map480iaray.constant36icm.map480icm algorithms479icm.mov480icm algorithms479icm.mov480icm algorithms477icm.mov480icm application refs480icm.ob482icm binary477icm.prf479icm commands122icm.rar480icm controls57icm.rs417icm flags31icm.rst416icm lumps59icm.seq420icm loops57icm.seq420icm loops57icm.tab418icm macros446icm.tot418Icm molecules60icm.vwt419icm able104icmCavityFinder390ICM-shell482, 492icmPlProfile479IcM-shell hitro26IcmSequence480icm.all477image anotation429icm.cfg479image center429icm.cff479imAGE_color110icm.cn477IMAGE_generateAlpha110icm.cn477IMAGE_generateAlpha110icm.cn477IMAGE_pereViewer110icm.cn477IMAGE_pereViewer110icm.cn477IMAGE_pereViewer	hydroxylation	448	icm.hdt	479
Iarray480icm.iar479Iarray reverse480icm.map480iarray.constant36icm.map480icm algorithms479icm.mov480icm application refs480icm.ob482icm binary477icm.prf479icm commands122icm.rar480icm controls57icm.rs417icm flags31icm.rst416icm functions291icm.se420icm loops57icm.se420icm loops57icm.tot418icm doubes22icm.tot418icm macros446icm.tot418icm molecules60icm.vwt419icm object file482, 492icmPhTPofile479icm.ashell objects26if184icm.ashell objects26if184icm.ash477image anotation429icm.cfg479image anotation429icm.cfg479image anotation429icm.cfg479image anotation429icm.cn477image anotation429icm.cn477image anotation429icm.cn477image high quality428icm.cn477iMAGE.generateAlpha110icm.cn477iMAGE.generateAlpha110icm.cn477iMAGE.generateAlpha110icm.cn477iMAGE				
Iarray reverse $480$ icm.map $480$ iarray.constant $36$ icm.mat $479$ icm algorithms $479$ icm.mov $480$ icm algorithms $479$ icm.mov $480$ icm application refs $480$ icm.ob $482$ icm binary $477$ icm.ob $482$ icm branching $58$ icm.rar $480$ icm commands $122$ icm.rar $482$ icm controls $57$ icm.rs $417$ icm flags $31$ icm.rs $416$ icm functions $291$ icm.se $420$ icm learning $426$ icm.seq $420$ icm loops $57$ icm.tab $418$ icm macros $446$ icm.tot $418$ icm molecules $60$ icm.vwt $419$ icm object file $482$ icm94 $479$ icm.able $104$ icmCavityFinder $390$ ICM-shell intro $26$ IcmSequence $480$ icm.shell objects $26$ if $184$ icm.all $477$ IMAGE $110$ icm.all $477$ Image annotation $429$ icm.bt $410$ image center $429$ icm.cfg $479$ IMAGE.color $110$ icm.cn $477$ IMAGE.color $110$ icm.cn $477$ IMAGE.color $110$ icm.cn $477$ IMAGE.color $110$ icm.cn $477$ IMAGE.color $110$ icm.cn $479$ IMAGE.	•			
iaray.constant36icm.mat479icm algorithms479icm.mov480icm application refs480icm.ob482icm binary477icm.prf479icm binary58icm.rar480icm commands122icm.res482icm commands122icm.res482icm controls57icm.res417icm flags31icm.res420icm functions291icm.ser420icm loops57icm.seq420icm loops57icm.seq420icm loops57icm.seq420icm loops57icm.seq420icm loops57icm.seq420icm loops57icm.tab418ICM modules22icm.var416icm object file482icm94479icm table104icmCavityFinder390ICM-shell intro26IcmSequence480icm.shell objects26if184icm.ali477IMAGE110icm.fg479IMAGE.color110icm.cn479IMAGE.color110icm.cnf477IMAGE.gammaCorrection110icm.cnf477IMAGE.gammaCorrection110icm.col418IMAGE.paerSize110icm.col418IMAGE.paerSize110	•			
icm algorithms $479$ icm.mov $480$ icm application refs $480$ icm.ob $482$ icm binary $477$ icm.ob $482$ icm binary $477$ icm.opf $479$ icm branching $58$ icm.rar $480$ icm commands $122$ icm.res $482$ icm controls $57$ icm.rs $417$ icm flags $31$ icm.rs $416$ icm functions $291$ icm.sar $479$ icm learning $426$ icm.seq $420$ icm learning $426$ icm.seq $420$ icm loops $57$ icm.tab $418$ ICM modules $22$ icm.var $416$ icm macros $446$ icm.var $416$ icm ble $104$ icmCavityFinder $390$ ICM-shell $482, 492$ icmPmProfile $479$ icm-shell objects $26$ if $184$ icm.ali $477$ IMAGE $110$ icm.shell objects $26$ if $184$ icm.bt $410$ image annotation $429$ icm.bt $410$ image center $429$ icm.cfg $479$ IMAGE.conpress $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cn $479$ IMAGE.generateAlpha $110$ icm.cn $479$ IMAGE.generateAlpha $110$ icm.cn	•		1	
icm application refs $480$ icm.ob $482$ icm binary $477$ icm.ob $482$ icm binary $477$ icm.prf $479$ icm branching $58$ icm.rar $480$ icm commands $122$ icm.res $482$ icm controls $57$ icm.res $412$ icm flags $31$ icm.rst $416$ icm flags $31$ icm.sar $479$ icm flags $59$ icm.se $420$ icm learning $426$ icm.seq $420$ icm loops $57$ icm.tab $418$ icm nocos $446$ icm.var $416$ icm modules $22$ icm.var $416$ icm molecules $60$ icm.var $416$ icm bject file $482$ , $492$ icmPufforfile $479$ icm-shell objects $26$ if $184$ icm-shell objects $26$ if $184$ icm.ali $477$ image $429$ icm.bt $410$ image annotation $429$ icm.bt $410$ image center $429$ icm.cfg $479$ IMAGE.color $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnf $479$ IMAGE.generateAlpha $110$ icm.cnf $479$ IMAGE.generateAlpha $110$ icm.cnf $479$ IMAGE.generateAlpha $110$ icm.	-			
icm binary $477$ icm.prf $479$ icm branching58icm.rar $480$ icm controls57icm.res $482$ icm controls57icm.res $417$ icm flags31icm.rs $416$ icm functions291icm.sar $420$ icm learning $426$ icm.seq $420$ icm loops57icm.seq $420$ icm loops57icm.tab $418$ icm macros $446$ icm.tot $418$ icm molecules60icm.var $416$ icm object file $482$ icm94 $479$ icm table104icmCavityFinder $390$ ICM-shell $482, 492$ icmSequence $480$ icm-shell objects $26$ if $184$ icm.ali $477$ image annotation $429$ icm.bt $410$ image center $429$ icm.cfg $479$ image center $429$ icm.cfg $479$ image figh quality $428$ icm.cn $477$ IMAGE.color $110$ icm.cn $477$ IMAGE.gammaCorrection $110$ <	•			
icm branching $58$ icm.rar $480$ icm commands122icm.res $482$ icm controls $57$ icm.res $417$ icm flags $31$ icm.rs $416$ icm functions $291$ icm.sar $479$ icm jumps $59$ icm.se $420$ icm learning $426$ icm.seq $420$ icm loops $57$ icm.tab $418$ icm macros $446$ icm.tot $418$ icm molecules $60$ icm.var $416$ icm object file $482$ icm94 $479$ icm table $104$ icmCavityFinder $390$ ICM-shell $482, 492$ icmPhrofile $479$ icm-shell objects $26$ if $184$ icm.all $477$ IMAGE $110$ icm.all $477$ image $429$ icm.bt $410$ image center $429$ icm.cfg $479$ IMAGE.color $110$ icm.cn $477$ IMAGE.generateAlpha $100$ icm.cn $477$ IMAGE.generateAlpha $100$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cod $410$ IMAGE.paperSize $110$ icm.cod $418$ IMAGE.paperSize $110$ icm.col $418$ IMAGE.paperSize $110$	**			-
icm commands122icm.res482icm controls57icm.rs417icm flags31icm.rs416icm functions291icm.sar479icm jumps59icm.se420icm learning426icm.seq420icm loops57icm.tab418icm macros446icm.tot418ICM modules22icm.var416icm molecules60icm.vwt419icm object file482icm94479icm table104icmCavityFinder390ICM-shell482, 492icmPmProfile479Icm-shell intro26IcmSequence480icm.all477IMAGE110icm.dl479image annotation429icm.bt410image annotation429icm.cn479IMAGE.color110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.color110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.generateAlpha110icm.cn479 </td <td>-</td> <td></td> <td>-</td> <td></td>	-		-	
icm controls $57$ icm.rs $417$ icm flags $31$ icm.rst $416$ icm functions $291$ icm.sar $479$ icm jumps $59$ icm.se $420$ icm learning $426$ icm.seq $420$ icm loops $57$ icm.tab $418$ icm macros $446$ icm.tot $418$ ICM modules $22$ icm.var $416$ icm molecules $60$ icm.vwt $419$ icm object file $482$ icm94 $479$ icm table $104$ icmCavityFinder $390$ ICM-shell $482, 492$ icmPmProfile $479$ ICM-shell intro $26$ IcmSequence $480$ icm-shell objects $26$ if $184$ icm.ali $477$ IMAGE $110$ icm.bt $410$ image annotation $429$ icm.cfg $479$ image high quality $428$ icm.cr $479$ IMAGE.color $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cn $477$ IMAGE.lineWidth $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cn $477$ IMAGE.lineWidth $110$ icm.cn $479$ IMAGE.generateAlpha $110$ icm.cn $479$ IMAGE.generateAlpha $110$ icm.cn $479$ IMAGE.generateAlpha $110$ icm.cn $479$ IMAGE.generateAlpha $110$ <td>-</td> <td></td> <td></td> <td></td>	-			
icm functions291icm.sar479icm jumps59icm.se420icm learning426icm.seq420icm loops57icm.tab418icm nacros446icm.tot418ICM modules22icm.var416icm molecules60icm.vwt419icm object file482icm94479icm table104icmCavityFinder390ICM-shell482, 492icmPmfProfile479ICM-shell intro26IcmSequence480icm-shell objects26if110icm.ali477IMAGE110icm.bbt410image annotation429icm.cfg479image center429icm.crn477IMAGE.color110icm.cn477IMAGE.gammaCorrection110icm.cn477IMAGE.gammaCorrection110icm.cn477IMAGE.gammaCorrection110icm.cn477IMAGE.gammaCorrection110icm.cnf477IMAGE.gammaCorrection110icm.cod410IMAGE.paperSize110icm.cod418IMAGE.paperSize110icm.col418IMAGE.paperSize110		57		417
icm functions291icm.sar479icm jumps59icm.se420icm learning426icm.seq420icm loops57icm.tab418icm nacros446icm.tot418ICM modules22icm.var416icm molecules60icm.vwt419icm object file482icm94479icm table104icmCavityFinder390ICM-shell482, 492icmPmfProfile479ICM-shell intro26IcmSequence480icm-shell objects26if110icm.ali477IMAGE110icm.bbt410image annotation429icm.cfg479image center429icm.crn477IMAGE.color110icm.cn477IMAGE.gammaCorrection110icm.cn477IMAGE.gammaCorrection110icm.cn477IMAGE.gammaCorrection110icm.cn477IMAGE.gammaCorrection110icm.cnf477IMAGE.gammaCorrection110icm.cod410IMAGE.paperSize110icm.cod418IMAGE.paperSize110icm.col418IMAGE.paperSize110	icm flags	31	icm.rst	416
icm jumps59icm.se420icm learning426icm.seq420icm loops57icm.tab418icm macros446icm.tot418ICM modules22icm.var416icm molecules60icm.vwt419icm object file482icm94479icm table104icmCavityFinder390ICM-shell482, 492icmPmfProfile479ICM-shell intro26IcmSequence480icm-shell objects26if184icm.ali477IMAGE110icm.ali477image annotation429icm.bt410image center429icm.cfg479IMAGE.color110icm.cn477IMAGE.generateAlpha110icm.cn477IMAGE.generateAlpha110icm.cnf477IMAGE.generateAlpha110icm.cnf477IMAGE.generateAlpha110icm.cnf477IMAGE.generateAlpha110icm.cnf479IMAGE.peperSize110icm.col418IMAGE.peperSize110icm.col418IMAGE.peperSize110	•		icm.sar	
icm learning $426$ icm.seq $420$ icm loops $57$ icm.tab $418$ icm macros $446$ icm.tot $418$ ICM modules $22$ icm.var $416$ icm molecules $60$ icm.vwt $419$ icm object file $482$ icm94 $479$ icm table $104$ icmCavityFinder $390$ ICM-shell $482, 492$ icmPmfProfile $479$ ICM-shell intro $26$ IcmSequence $480$ icm-shell objects $26$ if $184$ icm.ali $477$ IMAGE $110$ icm.all $477$ image annotation $429$ icm.bt $410$ image center $429$ icm.cfg $479$ IMAGE.color $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cn $477$ IMAGE.generateAlpha $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnd $477$ IMAGE.generateAlpha $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnd $477$ IMAGE.paperSize $110$ icm.cod $410$ IMAGE.paperSize $110$ icm.col $418$ IMAGE.paperSize $110$		59	icm.se	420
icm loops $57$ icm.tab $418$ icm macros $446$ icm.tot $418$ ICM modules $22$ icm.var $416$ icm molecules $60$ icm.var $416$ icm object file $482$ icm94 $479$ icm table $104$ icmCavityFinder $390$ ICM-shell $482, 492$ icmPmfProfile $479$ icm-shell objects $26$ IcmSequence $480$ icm-shell objects $26$ if $184$ icm.ali $477$ IMAGE $110$ icm.ali $477$ image annotation $429$ icm.bt $410$ image center $429$ icm.cfg $479$ IMAGE.color $110$ icm.cnn $477$ IMAGE.color $110$ icm.cnn $477$ IMAGE.gammaCorrection $110$ icm.cnf $477$ IMAGE.gammaCorrection $110$ icm.cnd $477$ IMAGE.gammaCorrection $110$ icm.cnd $477$ IMAGE.gammaCorrection $110$ icm.cnd $477$ IMAGE.gammaCorrection $110$ icm.cnd $477$ IMAGE.paperSize $110$ icm.col $418$ IMAGE.paperSize $110$	v 1	426	_	420
ICM modules22icm.var416icm molecules $60$ icm.vwt419icm object file $482$ icm94479icm table $104$ icmCavityFinder $390$ ICM-shell $482, 492$ icmPmfProfile $479$ ICM-shell intro $26$ IcmSequence $480$ icm-shell objects $26$ if $184$ icm.ali $477$ IMAGE $110$ icm.all $477$ image annotation $429$ icm.bbt $410$ image center $429$ icm.cfg $479$ IMAGE.color $110$ icm.cmp $477$ IMAGE.gammaCorrection $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cod $410$ IMAGE.orientation $110$ icm.col $418$ IMAGE.paperSize $110$ icm.col $479$ IMAGE.previewer $110$	U	57	icm.tab	418
icm molecules $60$ icm.vwt $419$ icm object file $482$ icm94 $479$ icm table $104$ icmCavityFinder $390$ ICM-shell $482, 492$ icmPmfProfile $479$ ICM-shell intro $26$ IcmSequence $480$ icm-shell objects $26$ if $184$ icm.ali $477$ IMAGE $110$ icm.all $477$ image annotation $429$ icm.bbt $410$ image center $429$ icm.cfg $479$ IMAGE.color $110$ icm.cmp $477$ IMAGE.color $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnd $410$ IMAGE.orientation $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cod $410$ IMAGE.orientation $110$ icm.cod $418$ IMAGE.paperSize $110$	icm macros	446	icm.tot	418
icm object file482icm94479icm table104icmCavityFinder390ICM-shell482, 492icmPmfProfile479ICM-shell intro26IcmSequence480icm-shell objects26if184icm.ali477IMAGE110icm.all477image429icm.bbt410image annotation429icm.cfg479image center429icm.crfg479IMAGE.color110icm.cnn477IMAGE.gammaCorrection110icm.cnf477IMAGE.gammaCorrection110icm.cnf477IMAGE.gammaCorrection110icm.cnd477IMAGE.gammaCorrection110icm.cnt477IMAGE.gameacorrection110icm.cnd477IMAGE.gameacorrection110icm.cnt479IMAGE.paperSize110icm.cod410IMAGE.paperSize110icm.col418IMAGE.paperSize110	ICM modules	22	icm.var	416
icm table $104$ icmCavityFinder $390$ ICM-shell $482, 492$ icmPmfProfile $479$ ICM-shell intro $26$ IcmSequence $480$ icm-shell objects $26$ if $184$ icm.ali $477$ IMAGE $110$ icm.all $477$ image $429$ icm.bbt $410$ image annotation $429$ icm.cfg $479$ image center $429$ icm.cnf $479$ IMAGE.color $110$ icm.cnn $477$ IMAGE.gammaCorrection $110$ icm.cnf $477$ IMAGE.generateAlpha $110$ icm.cnt $477$ IMAGE.lineWidth $110$ icm.cod $410$ IMAGE.orientation $110$ icm.col $418$ IMAGE.paperSize $110$ icm.col $479$ IMAGE.previewer $110$	icm molecules	60	icm.vwt	419
ICM-shell482,492icmPmfProfile479ICM-shell intro26IcmSequence480icm-shell objects26if184icm.ali477IMAGE110icm.ali477image429icm.bbt410image annotation429icm.bbt410image center429icm.cfg479image high quality428icm.crn479IMAGE.color110icm.cmp477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnf477IMAGE.lineWidth110icm.col418IMAGE.paperSize110	icm object file	482	icm94	479
ICM-shell intro26IcmSequence480icm-shell objects26if184icm.ali477IMAGE110icm.all477image429icm.bbt410image annotation429icm.bst410image center429icm.cfg479image high quality428icm.clr479IMAGE.color110icm.cmp477IMAGE.generateAlpha110icm.cnf477IMAGE.generateAlpha110icm.cnd477IMAGE.lineWidth110icm.cod418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm table	104	icmCavityFinder	390
icm-shell objects26if184icm.ali477IMAGE110icm.all477image429icm.bbt410image annotation429icm.bst410image center429icm.cfg479image high quality428icm.clr479IMAGE.color110icm.cmp477IMAGE.compress110icm.cnf477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnd477IMAGE.lineWidth110icm.cod410IMAGE.orientation110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	ICM-shell	482, 492	icmPmfProfile	479
icm.ali477IMAGE110icm.all477image429icm.blt410image annotation429icm.bst410image center429icm.cfg479image high quality428icm.clr479IMAGE.color110icm.cmp479IMAGE.compress110icm.cnf477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnd477IMAGE.lineWidth110icm.cnd477IMAGE.orientation110icm.cod410IMAGE.orientation110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	ICM-shell intro	26	IcmSequence	480
icm.all477image429icm.bbt410image annotation429icm.bst410image center429icm.cfg479image high quality428icm.clr479IMAGE.color110icm.cmp479IMAGE.compress110icm.cnf477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnd477IMAGE.lineWidth110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm-shell objects	26	if	184
icm.bbt410image annotation429icm.bst410image center429icm.cfg479image high quality428icm.clr479IMAGE.color110icm.cmp479IMAGE.compress110icm.cn477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnt477IMAGE.lineWidth110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.ali	477	IMAGE	110
icm.bst410image center429icm.cfg479image high quality428icm.clr479IMAGE.color110icm.cmp479IMAGE.compress110icm.cn477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnt477IMAGE.lineWidth110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.all	477	image	429
icm.cfg479image high quality428icm.clr479IMAGE.color110icm.cmp479IMAGE.compress110icm.cn477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnt477IMAGE.lineWidth110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.bbt	410	image annotation	429
icm.clr479IMAGE.color110icm.cmp479IMAGE.compress110icm.cn477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnf477IMAGE.lineWidth110icm.cnd410IMAGE.orientation110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.bst	410	image center	429
icm.cmp479IMAGE.compress110icm.cn477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnt477IMAGE.lineWidth110icm.cod410IMAGE.orientation110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.cfg	479	image high quality	428
icm.cn477IMAGE.gammaCorrection110icm.cnf477IMAGE.generateAlpha110icm.cnt477IMAGE.lineWidth110icm.cod410IMAGE.orientation110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.clr	479		110
icm.cnf477IMAGE.generateAlpha110icm.cnt477IMAGE.lineWidth110icm.cod410IMAGE.orientation110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.cmp	479		110
icm.cnt477IMAGE.lineWidth110icm.cod410IMAGE.orientation110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.cn	477	•	110
icm.cod410IMAGE.orientation110icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.cnf	477	<b>e</b> 1	110
icm.col418IMAGE.paperSize110icm.gro479IMAGE.previewer110	icm.cnt	477	IMAGE.lineWidth	
icm.gro 479 IMAGE.previewer 110				
		-		
icm.hbt 479 IMAGE.previewResolution 110	-			110
	icm.hbt	479	IMAGE.previewResolution	110

IMAGE.printerDPI	[	110	kly96	477
IMAGE.quality		110	kyt82	477
IMAGE.rgb2bw		110	label	166
IMAGE.scale		110	Label	427
IMAGE.stereoAng	le	110	label	492
IMAGE.stereoBase		110	Length	477
IMAGE.stereoText		110	LIBRARY	113
increment charge		239	lid00	480
Index		457	ligand binding	480
index expressions		37	ligand docking	467,468
Indexx		460	ligand docking intro	17
Insertion		330	ligand setting	468
Integer		482	limits	479
integer		492	LinearFit	477
integer array consta	ants	36	lineWidth	75
integer constant		36	Link	438
integer conversion		56	link internal variables of molecular object	184
integer shell variab	les	65	link sequences to 3D objects	185
Integral		482	link to alignment	185
interface residues		479	link variable	184
interface torsions		480	list	185
interface view		480	list database	186
internal coordinate	file	416	literature	477
internal coordinates	5	47	load	186
Interrupt		427	load conf	186
interruptAction		98	load frame	187
Introduction		1	load solution	187
inverting array orde	er	350, 480	Log	438
iProc		66	logarithm	438
isa98		480	logical	492
i_out		66	logical constant	36
I_out		120	logical operations	54
J-coupling		61	logical variables	81
jcp99		480	long axes	438
kab83		477	loop database rebuilding	286
keep		184	loop modeling intro	16
key mapping		479	l_antiAlias	81

l_autoLink	81	make disulfide bond	190
l_bpmc	82	make drestraint	191
1 breakRibbon	82	make factor	191
_ l_bufferedOutput	82	make grob image	193
l_bug	82	make grob map	192
l_caseSensitivity	82	make grob matrix	193
l_commands	82	make grob potential	194
l_confirm	83	make grob skin	194
l_easyRotate	83	make key	195
l_info	83	make map	196
1_minRedraw	83	make map factor	196
l_neutralAcids	83	make map potential	197
l_out	84	make peptide bond	198
l_print	84	make sequence	199
l_readMolArom	84	make tree	199
l_showAccessibility	84	make unique	200
l_showMC	85	makeIndexChemDb	400
l_showMinSteps	85	makeIndexSwiss	477
l_showSites	85	makePdbFromStereo	477
l_showSpecialChar	85	makeSimpleDockObj	407
l_showSstructure	85	makeSimpleModel	407
l_showTerms	85	making Swissprot index	284
l_showWater	85	manual style	24
l_warn	86	Мар	479
l_wrapLine	86	map	492
l_writeStartObjMC	86	map averaging	438
l_xrUseHydrogen	86	map calculation	196
maa89	479	map file	480
macro	188,492	map fitting	94
mai97	480	map transformations	438
mai98	480	map.contouring	194
main concepts	457	mapping properties to sequence	482
main ICM references	479	maps and factors	460
make	189	mapSigmaLevel	75
make bond	189	Mass	479
make bond chain	189	Matrix	479
make boundary	190	matrix	492

Max	446	mncallsMC	68
maxColorPotential	66	mnconf	68
MaxHKL	480	mnhighEnergy	68
maxMemory	67	mnreject	68
mcBell	75	mnSolutions	67
mcJump	75	mnvisits	69
mcl79	477	Mod	457
mcShake	75	model reliability	479
mcStep	76	modify	203
Mean	480	modify as_	204
menu	202	modify as_ library	203
menu script	426	modify rs_	203
merge arrays to table	180	mol	255
merge objects	210	Mol	460
merge pdb	477	mol to icm	149
merge sarray into string	482	mol2	492
merge stacks	446	molecular manipulations	446
Method	97	molecular modifications	448
methylation	448	molecular surface	492
mf	61	molecular views	13
mfMethod	98	molecule create	448
mfWeight	76	molecule intermediate	479
mimel	492	molecule properties	243
mimelDepth	76	molecule rotation	428
mimelMolDensity	76	molecule.create	29
Min	480	molecules intro	39
minimize	67,200	molecules sort/reorder	269
minimize cartesian	201	molecules.attributes	255
minimize loop	202	molecules.selecting	42
minimize tether	67,202	mom75	477
minimize.drop	73	Money	457
minimizeMethod	98	montecarlo	68,69
minTetherWindow	67	montecarlo command	205
mkUniqPdbSequences	477	montecarlo fast	205
mmff	492	montecarlo local	69
mmff type	380	montecarlo trajectory	164
mmff.show atom types	259	more	492
mncalls	67	mouse controls	32

move	210	OBJECT	113
move atoms	237	object properties	243
move ms_molecule	210	object.assign comment	240
move os	210	object.attributes	255
movie	187, 492	object.copy	150
movie file	480	object.translate	274
movie molecular simulation	482	objects merge	477
movie smoothing	164	objects sort/reorder	269
movie zooming	480	objects.selecting	41
multiple alignment	123	Occupancy	477
multiple sequence alignment intro	20	on–line help	183
mutate DNA or RNA	203	only	492
mutate residue	203	or	54
mute	492	out-of plane energy	61
m_ga	61	output	66
M_out	120	packing density	482
Name	460	parallelization	66,69
Namex	340	pat98	480
nee70	477	Path	477
nem83	477	Pattern	438
nem92	477	pattern matching	55,492
Newick tree format	480	pattern search	170
Next	340,482	pause	211
nice	400	pdb	492
nLocalDeformVar	69	pdb files	419
NOE	61	pdb merge	477
Nof	482	pdb sequence generation	440
non-redundant	440	pdb waters by number	42
Norm	427	pdbDirStyle	99
normal distribution	460	pea88	477
notational conventions	24	peptide	448
nProc	69	peptide cyclization	448
nSsearchStep	69	peptide docking	479
number of elements	482	peptide folding intro	16
number of occurencies	482	personal gui controls	482
nvis	480	personal setup	482
Obj	477	phosphorylation	448

Pi	438	pocket	480
pK shift	480	Potential	438
PLOT	114	pov–ray	290
plot	211	Power	479
plot 3D 2Dfunction	479	predictSeq	479
plot 3D shape	479	preference	93
plot area	213	prepSwiss	480
plot histogram	477	previous atom	482
plot how to	477	principal axes	438
plot simple	477	principal component analysis	482
PLOT.box	114	print	215
PLOT.color	114	print image	216
PLOT.font	114	print to string	271
PLOT.fontSize	114	printf	215
PLOT.labelFont	114	printFast	480
PLOT.lineWidth	114	printMatrix	480
PLOT.logo	114	printPostScript	480
PLOT.markSize	114	printTorsions	480
PLOT.numberOffset	114	Probability	479
PLOT.orientation	114	Profile	457
PLOT.rainbowStyle	114	profile	492
PLOT.seriesLabels	114	program overview	13
PLOT.Yratio	114	projected alignment	56
plot2DSeq	477	property	246
plotBestEnergies	479	prosite	492
plotCluster	479	prosite pattern	492
plotFlexibility	479	protein docking intro	17
plotMatrix	479	protein grid docking	482
plotOldEnergy	479	Putenv	457
plotRama	479	quit	216
plotRose	479	radii.electrostatic	457
plotSeqDotMatrix	477	radii.van der Waals	457
plotSeqDotMatrix2	477	Radius	457
plotSeqProperty	479	ramachandran how to	479
plotting van der Waals	479	random	102
pmf	61,98	Random	460
pmf residue profile	479	random array	460
png	492	randomize	69,216

randomize atoms	217	read mol	83,226
randomize in range	216	read mol2	227
randomize molecules	217	read movie	227
randomize torsions	216	read movie write	227
randomSeed	69	read object	228
Rarray	460	read pdb	228
rarray	350	read pdb sequence	230
Rarray reverse	350	read profile	231
Rarray sequence projection	350	read prosite	231
Rarray.alignment projection	482	read rarray	231
Rarray.property assignment	482	read sarray	231
rdBlastOutput	482	read segment	231
rdSeqTab	482	read sequence	232
read	217	read sequence database	232
read alignment	221	read stack	232
read all	218	read string	232
read color	222	read table	233
read column	233	read unix	220
read comp_matrix	222	read unix cat	221
read conf	222	read variable	233
read csd	222	read view	233
read database	223	read vrestraint	233
read drestraint	223	read vrestraint type	233
read drestraint type	223	readMolNames	121
read factor	223	readPdbList	482
read FILTER	218	Real	482
read from file	217	real	36
read from string	218	real array constants	36
read ftp	220	real constant	36
read grob	224	real shell variables	71
read http	220	real space refinement	61
read iarray	224	rebel	66
read index	224	REBEL	479
read index table	219	rebel	492
read library	225	Reference Guide	31
read library mmff	225	references	477
read map	225	refineModel	480
read matrix	226	refresh view	159

regul	482	reverse complement	482
regular expressions	492	reversing order	480
regularization	67,492	Rfactor	479
regularization procedure	447	Rfree	479
rejectAction	99	rgb	492
relational database	438	ribbon	492
release notes	1	ribbonColorStyle	100
Remainder	427	ribbonStyle	100
remarkObj	482	Rmsd	479
rename	234	Rot	446
rename.atom	235	rotate	235
rename.molecule	235	rotate grob	236
rename.residue	235	rotate object	236
reorder alignment sequences	278	rotate view	236
Replace	477	rounding a real	480
reproducible randomness	69	rs	61
reroot	150	rsWeight	77
Res	438	run script	136
reserved names	119	r_2out	77
residue	492	r_out	76
residue contact areas	298	R_out	120
residue cursor	140	s–s bond	448
residue field	243	sai87	477
residue library file	482	Sarray	480
residue property averaging	477	Sarray reverse	480
residue ranges	50	sarray reverse	480
residue renumbering	123	save print	429
residue selection as string	460	sch00	480
residue selection function	438	sch99	480
residue user field	482	Score	457
residues.selecting	43	script	492
resLabelShift	77	script.image generation	159
resLabelStyle	99	search pdb headers	479
Resolution	479	search prosite	438
restraints	61	search protein fragment	438
restraints.torsion	251	search protein topology	440
return	235	search sstructure database	477
reverse	480	search.sequence pattern	174

searches and alignments	438	selection.displayed atoms	45
searchObjSegment	406	selection.functions	50
searchPatternDb	482	selection.molecules	42
searchPatternPdb	406	selection.objects	41
searchSeqDb	406	selection.output	122
searchSeqFullPdb	406	selection.residues	43
searchSeqPdb	406	selection.torsions	47
searchSeqProsite	407	selection.variables	47
searchSeqSwiss	407	selection.waters	42
second moments	438	selections in molecular objects	39
secondary structure derivation from 3D	128	selectMinGrad	77
segment	492	selectSphereRadius	77
segMinLength	70	Sequence	360
Select	460	sequence	492
selected atoms	45	sequence alignment intro	20
selecting by b-factor	460	sequence analysis intro	19
selecting by x y z	460	sequence assembly	178
selection by tether	45	sequence dotplot	19
selection by tether destination	45	sequence from pdb	199
selection level	39,380	sequence pattern	492
selection multiplication	56	sequence positional weights	237
selection precedence	56	sequence redundancy removal	178
selection transfer	460	sequence search	170
selection type	39	sequence to alignment transfer	482
selection variable	121	sequence type	492
selection.atoms	45	Sequence(dna reverse)	482
selection.atoms by code	45	sequence-alignment mapping	480
selection.atoms by gradient	45	sequence.output format	70
selection.atoms with alternatives	45	sequenceBlock	70
selection.by alignment consensus	43	sequenceLine	70
selection.by residues feature	43	sequences	438
selection.by site type	43	set	236

set area	237	set type sequence	250
set atom	237	set variable grid	253
set bfactor	238	set view	251
set bond type	238	set vrestraint	251
set charge	239	set vw radii	250
set charge formal	239	set window	254
set charge mmff	240	setResLabel	407
set comment	240	sf	61
set comment sequence	241	shell	482
set comp_matrix	241	shell warning message	479
set current map	245	shineStyle	101
set current object	245	shininess	78
set directory	241	show	254,266
set drestraint	241	show aliases	256
set drestraint type	242	show alignment	257
set electrostatic radii	250	show area	257
set field	243	show atom type	259
set font	243	show atoms	258
set grob coordinates	244	show clash	260
set key	244	show color	260
set label	245	show column	261
set label distance	245	show comp_matrix	261
set molecular variables	253	show drestraint	261
set occupancy	246	show drestraint type	262
set plane	246	show energy	262
set property	246	show gradient	262
set site	242	show hbond	263
set site residue	243	show hbond exact	263
set sstructure backbone	246	show html	263
set sstructure to sequence	247	show iarray	263
set stack energy	248	show integer	264
set symmetry group	248	show key	255
set symmetry to a torsion	248	show label	264
set table	248	show library	264
set terms	249	show link	264
set tether	249	show logical	264
set type	250	show map	255
set type mmff	250	show mol	265

show molecule265sln492show molecules255SLN notation460show object265Smiles492show phb265smiles492show preferences265Smooth71, 427, 477show residue type266sort accessible area70show segment266sort arrays269show sequence266sort objects269show sequence266sort objects269show sequence266sort objects269show shell variable254sort objects269show tabk267sort stack269show table267sort stack269show table267sort stack269show table267sort stack269show table267split270show table267split270show tethers267split270show version267split able271show version268split table271show version268split able271show verstraint type268sprintf271show vrestraint type268sprintf272sign482search69, 272sign482search69, 272sin4124Srmsd446simulations intro16ss61Sin492Startcure370Site492 <th>show mol2</th> <th>265</th> <th>skin intro</th> <th>13</th>	show mol2	265	skin intro	13
show object265Smiles427show pdb265smiles492show preferences265Smooth71, 427, 477show residue266solver accessible area70show residue type266sort arrays269show seguence266sort arrays269show sequence266sort objects269show sequence254sort objects269show site254sort objects269show stack267sort table269show stack267sort stack269show table as database261Sphere479show tethers267split446show version267split table270show version267split table270show version268split table271show volume268split table271show versinints268split table271show versinints268split438shr73477Sqtt479Sign482sri98480simulations intro16ss61Sin427ssearchStep78SiTE116stack bin size46SITE.labelOffset116stack bin size446SITE.labelWrap116stack smerge446SITE.labelWrap116stack smerge446SITE.labelWrap116stack smerge446<	show molecule	265	sln	492
show pdb265smiles492show preferences265Smooth71, 427, 477show residue266sort accessible area70show residue type266sort arrays269show segment266sort molecules269show sequence266sort objects269show shell variable254sort objects269show stack267sort stack269show table267sort stack269show table267sort stack269show table267sphere479show table267Split270show tethers267Split271show version267Split object271show volume268sprintf271show volume map268sprintf271show verstraint type268Sql438shr73477Sqrt479Sign482sri98480sim4124Srmsd446simulations intro16ss61Sin427ssearch69, 272Sinh427ssflreshold78Site492Structure370Site116ssearch68, 69, 102, 271, 477, 480, 486Site.alelOffset116stack bin size144Sitte.labelUyle116stack bin size446Site.alelOffset116stack bin size446Site.alelO	show molecules	255	SLN notation	460
show preferences265Smooth71, 427, 477show residue266solvent accessible area70show residue type266sort arrays269show seguence266sort molecules269show sequence266sort molecules269show site254sort objects269show site254sort stack269show site267sort table269show table267sort Seq407show table as database261Sphere479show tethers267split446show version267split object271show version268split table270show verstaint type268splitting selection50show verstraints268Sql438shr73477Sqrt479Sign482sri98480sim4124Srmsd446simulations intro16ss61Sin427ssearchStep78SiTE116ssign sstructure segment129Site427ssfructure370SiTE LabelOffset116stack bin size444STTE.labelStyle116stack sirege446STTE.labelStyle116stack sirege446STTE.labelStyle116stack sirege446STTE.labelStyle116stack sirege446STTE.labelStyle116stack s	show object	265	Smiles	427
show residue266solvent accessible area70show residue type266sort269show segment266sort arrays269show sequence266sort molecules269show shell variable254sort stack269show site254sort stack269show stack267sort table269show table267sort stack269show table as database261Sphere479show term267split270show version267split object271show volume268split table270show volume map268split table270show volume map268split table271show verstraint type268Sql438shr73477Sqrt479Sign482sri98446simulations intro16ss61Sin427scarchStep78SITE116ssign structure segment129Site427sThreshold78SiTE LabelOffset116stack bin size444SITE.labelStyle116stack sinege446SITE.labelStyle116stack sinege446SITE.labelStyle116stack sinege446SITE.labelWrap116stack sinege446SITE.labelStyle116stack sinege446SITE.labelStyle116stack sim	show pdb	265	smiles	492
show residue type266sort269show segment266sort arrays269show sequence266sort molecules269show shell variable254sort objects269show site254sort tack269show site267sort tack269show tack267sort table269show table267sort Seq407show table as database261Sphere479show term267split270show terms267split object271show version267split object271show version268split table270show volume268split gelection50show vrestraint type268sprintf271show verstraint type268Sql438show verstraint type268Sql438show verstraints268Sql438show verstraint type268Sql438sinf477Sqrt479Sign482srin98480simulations intro16ss61Sin427ssearchStep78SiTE116stack68, 69, 102, 271, 400, 486SiTE.labelOffset116stack bin size144SITE.labelStyle116stack bin size144SITE.labelStyle116stack bin size144SITE.labelStyle116stack smerge4	show preferences	265	Smooth	71, 427, 477
show segment266sort arrays269show sequence266sort molecules269show shell variable254sort objects269show site254sort stack269show site254sort stack269show stack267sort table269show table267sortSeq407show table as database261Sphere479show table as database267split246show tethers267split446show version267split table270show volume268split table270show volume map268split table271show volume map268split table271show volume map268split able271show vestraint type268split able271sima427sgreach69, 272<	show residue	266	solvent accessible area	70
show sequence         266         sort molecules         269           show shell variable         254         sort objects         269           show site         254         sort stack         269           show stack         267         sort stack         269           show table         267         sort sort stack         269           show table         267         sort sort sort able         269           show table as database         261         Sphere         479           show table as database         267         split         446           show version         267         split object         271           show version         267         split table         270           show version         268         split table         270           show volume map         268         split table         271           show vrestraint type         268         split table         271           show vrestraints         268         Sql         438           sh73         477         Sqrt         479           sign         482         sri98         480           sim4         124         Srmsd         410	show residue type	266	sort	269
show shell variable254sort objects269show site254sort stack269show site267sort table269show table267sort stack269show table as database261Sphere479show term267split446show version267Split446show version267split object271show volume268split table270show version268split table270show verstraint type268sprintf271show verstraint type268Sql438shr73477Sqrt479sign482sri98480sim4124Srmsd446simulations intro16ss61Sin427ssearchStep78SITE116ssign structure segment129Site422Structure370SITE.labelOffset116stack bin size444SITE.labelWrap116stack bin size446SITE.labelWrap116stack bin size446<	show segment	266	sort arrays	269
show site $254$ sort stack $269$ show stack $267$ sort table $269$ show table $267$ sortSeq $407$ show table as database $261$ Sphere $479$ show term $267$ split $270$ show ters $267$ Split $446$ show version $267$ split object $271$ show volume $268$ split table $270$ show verstraint type $268$ spliting selection $50$ show vestraint type $268$ Sql $438$ shr73 $477$ Sqrt $479$ Sign $482$ sri98 $480$ sim4 $124$ Srmsd $446$ simulations intro $16$ ss $61$ Sin $482$ ssearch $69, 272$ Sinh $427$ ssearchStep $78$ SITE $116$ sign structure segment $129$ Site $427$ sstructure $370$ SITE.labelOffset $116$ stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle $116$ stack bin size $144$ SITE.labelStyle $116$ stack smerge $446$ SITE.labelWrap $116$ stack smerge $446$ SITE.labelStyle $116$ </td <td>show sequence</td> <td>266</td> <td>sort molecules</td> <td>269</td>	show sequence	266	sort molecules	269
show stack         267         sort table         269           show table         267         sortSeq         407           show table as database         261         Sphere         479           show term         267         split         270           show term         267         split         446           show version         267         split object         271           show volume         268         split table         270           show volume map         268         split table         271           show vrestraint type         268         spliting selection         50           show vrestraints         268         Split         438           shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           simulations intro         16         ss earch         69, 272           Sinh         427         ssearchStep         78           Site         427         ssfure structure segment         129           Site         492         Structure         370           SITE LabelOffset         116 <td>show shell variable</td> <td>254</td> <td>sort objects</td> <td>269</td>	show shell variable	254	sort objects	269
show table         267         sortSeq         407           show table as database         261         Sphere         479           show term         267         split         270           show tethers         267         Split         446           show version         267         split object         271           show volume         268         split table         270           show volume map         268         sprintf         271           show vrestraint type         268         sprintf         271           show vrestraints         268         Sqlt         438           shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           simulations intro         16         ss         69, 272           Sinh         482         ssearchStep         78           SITE         116         ssign structure segment         129           Site         492         Structure         370           SITE.labelOffset         116         stack         68, 69, 102, 271, 477, 480, 486           SITE.labelOffset         <	show site	254	sort stack	269
show table as database         261         Sphere         479           show term         267         split         270           show tethers         267         Split         446           show version         267         split object         271           show volume         268         split table         270           show volume map         268         spliting selection         50           show verstraint type         268         Split         38           show verstraints         268         Sql         438           show verstraints         268         Sql         438           show verstraints         268         Sql         479           show verstraints         268         Sql         438           show verstraints         268         Sql         438           shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           Sinn         427         ssearch         69, 272           Sinh         427         ssearchStep         78           SiTE         116         sstructure segment<	show stack	267	sort table	269
show term         267         split         270           show tethers         267         Split         446           show version         267         split object         271           show volume         268         split table         270           show volume map         268         split table         270           show vrestraint type         268         sprintf         271           show vrestraints         268         Sql         438           shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           simulations intro         16         ss         61           Sin         482         ssearch         69, 272           Sinh         427         ssearchStep         78           SITE         116         ssign structure segment         129           Site         492         Structure         370           SITE.labelOffset         116         ssWeight         78           SITE.labelOffset         116         stack bin size         144           SITE.labelVap         116         stack bi	show table	267	sortSeq	407
show tethers         267         Split         446           show version         267         split object         271           show volume         268         split table         270           show volume map         268         splitting selection         50           show vrestraint type         268         sprintf         271           show vrestraints         268         splitting selection         50           show vrestraints         268         Sql         438           shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           simulations intro         16         ss         61           Sin         482         ssearch         69, 272           Sinh         427         ssearchStep         78           SITE         116         ssign sstructure segment         129           Site         492         Sstructure         370           SITE.defSelect         116         sack         68, 69, 102, 271, 477, 480, 486           SITE.labelOffset         116         stack bin size         144           SITE.lab	show table as database	261	Sphere	479
show version267split object271show volume268split table270show volume map268splitting selection50show vrestraint type268sprintf271show vrestraints268Sql438shr73477Sqrt479Sign482sri98480sim4124Srmsd446simulations intro16ss61Sin482ssearch69,272Sinh427ssearchStep78SITE116ssign sstructure segment129Site427ssThreshold78site492Structure370SITE.labelOffset116stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle116stack bin size144SITE.labelWrap116stack smerge446SITE.showSeqSkip116stereo reconstruction479	show term	267	split	270
show volume         268         split table         270           show volume map         268         splitting selection         50           show vrestraint type         268         sprintf         271           show vrestraints         268         Sql         438           shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           simulations intro         16         ss         61           Sin         482         ssearch         69,272           Sinh         427         ssearchStep         78           SITE         116         ssign sstructure segment         129           Site         492         Structure         370           SITE.labelOffset         116         sseWeight         78           SITE.labelOffset         116         stack bin size         144           SITE.labelVap         116         stack bin size         144           SITE.labelWrap         116         stack merge         446           SITE.showSeqSkip         116         stacks merge         446	show tethers	267	Split	446
show volume map         268         splitting selection         50           show vrestraint type         268         sprintf         271           show vrestraints         268         Sql         438           shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           simulations intro         16         ss         61           Sin         482         ssearch         69, 272           Sinh         427         ssearchStep         78           SITE         116         ssign sstructure segment         129           Site         427         ssfreshold         78           SITE         116         ssign sstructure segment         129           Site         492         Structure         370           SITE.labelOffset         116         sack         68, 69, 102, 271, 477, 480, 486           SITE.labelOffset         116         stack bin size         144           SITE.labelVap         116         stack prese         144           SITE.labelWap         116         stacks merge         446           SITE.showSeqSk	show version	267	split object	271
show vrestraint type268sprintf271show vrestraints268Sql438shr73477Sqrt479Sign482sri98480sim4124Srmsd446simulations intro16ss61Sin482ssearch69, 272Sinh427ssearchStep78SITE116ssign sstructure segment129Site427ssThreshold78site492Structure370SITE.labelOffset116stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle116stack bin size144SITE.labelWrap116stack bin size446SITE.showSeqSkip116stereo reconstruction479	show volume	268	split table	270
show vrestraints         268         Sql         438           shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           simulations intro         16         ss         61           Sin         482         ssearch         69, 272           Sinh         427         ssearchStep         78           SITE         116         ssign sstructure segment         129           Site         427         ssThreshold         78           SITE         116         ssign sstructure segment         129           Site         492         Structure         370           SITE.defSelect         116         ssWeight         78           SITE.labelOffset         116         stack         68, 69, 102, 271, 477, 480, 486           SITE.labelVap         116         stack bin size         144           SITE.labelWrap         116         stack grapp         102           site.selecting residues by         43         stacks merge         446           SITE.showSeqSkip         116         stereo reconstruction         479	show volume map	268	splitting selection	50
shr73         477         Sqrt         479           Sign         482         sri98         480           sim4         124         Srmsd         446           simulations intro         16         ss         61           Sin         482         ssearch         69,272           Sinh         427         ssearchStep         78           SITE         116         ssign sstructure segment         129           Site         427         ssThreshold         78           SITE         116         ssign sstructure segment         129           Site         492         Structure         370           SITE.defSelect         116         ssWeight         78           SITE.labelOffset         116         stack         68,69,102,271, 477,480,486           SITE.labelOffset         116         stack bin size         144           SITE.labelWrap         116         stack jump         102           site.selecting residues by         43         stacks merge         446           SITE.showSeqSkip         116         stereo reconstruction         479	show vrestraint type	268	sprintf	271
Sign $482$ sri98 $480$ sim4124Srmsd $446$ simulations intro16ss $61$ Sin482ssearch $69, 272$ Sinh427ssearchStep $78$ SITE116ssign sstructure segment $129$ Site427ssThreshold $78$ site492Sstructure $370$ SITE.defSelect116ssWeight $78$ SITE.labelOffset116stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle116stack bin size $144$ SITE.labelWrap116stack smerge $446$ SITE.showSeqSkip116stereo reconstruction $479$	show vrestraints	268	Sql	438
sin4124Srmsd446simulations intro16ss61Sin482ssearch69,272Sinh427ssearchStep78SITE116ssign sstructure segment129Site427ssThreshold78site492Sstructure370SITE.defSelect116ssWeight78SITE.labelOffset116stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle116stack bin size144SITE.labelWrap116stacks merge446SITE.showSeqSkip116stereo reconstruction479	shr73	477	Sqrt	479
simulations intro16ss61Sin482ssearch69, 272Sinh427ssearchStep78SITE116ssign sstructure segment129Site427ssThreshold78site492Sstructure370SITE.defSelect116ssWeight78SITE.labelOffset116stack68, 69, 102, 271, 477, 480, 486SITE.labelStyle116stack bin size144SITE.labelWrap116stacks merge446SITE.showSeqSkip116stereo reconstruction479	Sign	482	sri98	480
Sin482ssearch69,272Sinh427ssearchStep78SITE116ssign sstructure segment129Site427ssThreshold78site492Sstructure370SITE.defSelect116ssWeight78SITE.labelOffset116stack68,69,102,271, 477,480,486SITE.labelStyle116stack bin size144SITE.labelWrap116stack smerge446SITE.showSeqSkip116stereo reconstruction479	sim4	124	Srmsd	446
Sinh427ssearchStep78SITE116ssign sstructure segment129Site427ssThreshold78site492Sstructure370SITE.defSelect116ssWeight78SITE.labelOffset116stack68, 69, 102, 271, 477, 480, 486SITE.labelStyle116stack bin size144SITE.labelWrap116stack smerge446SITE.showSeqSkip116stereo reconstruction479	simulations intro	16	SS	61
SITE116ssign sstructure segment129Site427ssThreshold78site492Sstructure370SITE.defSelect116ssWeight78SITE.labelOffset116stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle116stack bin size144SITE.labelWrap116stackjump102site.selecting residues by43stacks merge446SITE.showSeqSkip116stereo reconstruction479	Sin	482	ssearch	69,272
Site427ssThreshold78site492Sstructure370SITE.defSelect116ssWeight78SITE.labelOffset116stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle116stack bin size144SITE.labelWrap116stackjump102site.selecting residues by43stacks merge446SITE.showSeqSkip116stereo reconstruction479	Sinh	427	ssearchStep	78
site492Sstructure370SITE.defSelect116ssWeight78SITE.labelOffset116stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle116stack bin size144SITE.labelWrap116stackjump102site.selecting residues by43stacks merge446SITE.showSeqSkip116stereo reconstruction479	SITE	116	ssign sstructure segment	129
SITE.defSelect116ssWeight78SITE.labelOffset116stack $68, 69, 102, 271, 477, 480, 486$ SITE.labelStyle116stack bin size144SITE.labelWrap116stackjump102site.selecting residues by43stacks merge446SITE.showSeqSkip116stereo reconstruction479	Site	427	ssThreshold	78
SITE.labelOffset116stack68, 69, 102, 271, 477, 480, 486SITE.labelStyle116stack bin size144SITE.labelWrap116stackjump102site.selecting residues by43stacks merge446SITE.showSeqSkip116stereo reconstruction479	site	492	Sstructure	370
SITE.labelOfiset116stack477, 480, 486SITE.labelStyle116stack bin size144SITE.labelWrap116stackjump102site.selecting residues by43stacks merge446SITE.showSeqSkip116stereo reconstruction479	SITE.defSelect	116	ssWeight	78
SITE.labelWrap116stackjump102site.selecting residues by43stacks merge446SITE.showSeqSkip116stereo reconstruction479	SITE.labelOffset	116	stack	
site.selecting residues by43stacks merge446SITE.showSeqSkip116stereo reconstruction479	SITE.labelStyle	116	stack bin size	144
SITE.showSeqSkip 116 stereo reconstruction 479	SITE.labelWrap	116	stackjump	102
	site.selecting residues by	43	stacks merge	446
skin 492 sti99 480	SITE.showSeqSkip	116	stereo reconstruction	479
	skin	492	sti99	480

stick	492	svariable	492
store conf	271	swapping protein fragments	479
store torsion type	271	swissFields	120
str96	480	Symgroup	427
strain	45	symmetry group	66
String	460,480	system.reserved selections	122
string	56	s_blastdbDir	86
string addition	56	s_editor	87
string array constants	36	s_entryDelimiter	87
string comparisons	55	s_errorFormat	87
string constant	36	s_fieldDelimiter	87
string filtering	460	s_helpEngine	88
string inversion	480	s_icmhome	88
string variables	86	s_icmPrompt	88
string+number	56	s_imageViewer	89
String.chemical formula	460	s_inxDir	88
strip	272	s_labelHeader	89
structural alignment	125	s_lib	89
structural alignment optimization	169	s_logDir	89
structural superposition	61,125	s_out	90
structure analysis	477	S_out	120
structure comparison	480	s_pdbDir	90
structure structure	460	s_printCommand	90
subalignment to selection	438	s_projectDir	90
substring	480	s_prositeDat	91
substructure search	172	s_psViewer	91
sulfation	448	s_reslib	91
Sum	482	s_skipMessages	91
superimpose	273	s_tempDir	92
superimpose how to	477	s_translateString	92
surface	492	s_userDir	92
surface area	492	s_usrlib	92
surface point selection	479	s_webEntrezLink	92
surface tension	61	s_webViewer	92
surfaceAccuracy	70	s_xpdbDir	93
surfaceMethod	101	tab94	479
surfaceTension	78	table	246

Table	427	terms.tethers	61
table	510	terms.torsion	61
table creation	180	terms.van der Waals (vw)	61
table expression	510	terms.variable restraints	61
table subset	510	tether	279,510
Table(alignment)	477	tethers	61
Table(stack)	477	tha97	480
table.show html	277	then	273
Tan	438	thg94	477
Tanh	438	tho94	477
temperature	79	tif	511
Temperature	479	Time	446
tempLocal	78	to	61
Tensor	438	toa96	479
tensor product of two vectors	479	tolGrad	79
terminal font	479	Tolower	446
terminal window	181	tom00	480
terms	61	topology files	409
terms drestraints	61	topology search	175
terms.bond stretching	61	Torsion	446
terms.density correlation	61	torsion restraints	61
terms.disulfide bonds	61	Toupper	480
terms.electrostatic	61	Tr123	480
terms.entropy	61	Tr321	480
terms.grid electrostatic potential	61	Trace	457
terms.grid h-bonding	61	Trans	457
terms.grid hydrophobic energy	61	transform	274
terms.grid small probe vw energy	61	transformation	511
terms.grid vw energy	61	transformations and symmetry	457
terms.hydrogen bonding	61,74	translate	274
terms.local vw interactions (14)	61	transparent grob	162
terms.mean-force potential	61	Transpose	460
terms.phase angle bending	61	Trim	460
terms.R-factor	61	Turn	380
terms.surface energy	61	Туре	380

tz	61	Volume	477
tzMethod	101	volume	512
tzWeight	79	vrestraint	512
unclip	159	vrestraint file	417
undisplay	275	vrestraint type	512
undsCharge	407	vrestraint type file	416
unfix	275	vs_out	122
unique	512	VW	61
unique atomic order	200	vwCutoff	80
unique coloring	143	vwExpand	80
unique smiles	200	vwMethod	102
unix	276	vwSoftMaxEnergy	80
Unix	482	wait	276
updates	1	Warning	479
url string parsing	427	warning message	479
user-defined properties	243	warning suppression	86
users guide	482	water.dielectric constant	73
user_startup.icm	482	waterRadius	80
Value	427	wavefront format	193
van der Waals 1–4	61	Wavefront format	224
variable restraint	251	web	276
variable selection	47	web table	277
varLabelStyle	102	WEBAUTOLINK	118
vea91	477	webEntrezOption	103
Vector	427	WEBLINK	117
vector product	427	wei86	477
Vector.symmetry transformation	427	wei88	477
Version	427	wes92	477
vicinity	79	while	277
View	438	wil84	477
virtual	512	window averaging	477
virtual ligand screening	467, 468, 477	window width and height	438
virtual ligand screening intro	18	windowSize	71
visitsAction	102	wire	512
vls	467, 468, 477	wireBondSeparation	81
vls scoring	61	wireStyle	103

write	278	xrMethod
write alignment	278	xrWeight
write as table	279	xstick
write blast	284	Xyz
write column	279	yuj97
write database	279	zega
write drestraint	281	ZEGA intro
write drestraint type	282	zha92
write factor	282	zha99
write grob	282	zho98
write html	282	zho99
write iarray	279	_macro.icm file
write image	282	_startCheck
write index	284	_startup.icm
write library	285	
write map	285	
write matrix	279	
write model	286	
write mol	286	
write mol2	287	
write movie	164	
write object	287	
write object simple	288	
write pdb	288	
write postscript	288	
write pov	290	
write povray	290	
write rarray	279	
write sarray	279	
write segment	290	
write sequence	290	
write session	290	
write several array	279	
write stack	291	
write table	279	
write tethers	279	
write vs_var	291	
xr	61	