

Programming in MolTalk

Alexander V. Diemand

12th March 2004

<http://www.moltalk.org>

Contents

I MolTalk Library	4
1 Requisites	4
2 Class diagram	4
3 Structural classes	5
3.1 StructureFactory	5
3.2 Structure	5
3.3 ChainFactory	6
3.4 Chain	6
3.5 ResidueFactory	7
3.6 Residue	8
3.7 Atom	9
3.8 Coordinates	9
3.9 PairwiseStrxAlignment	10
3.10 Selection	10
4 Mathematical classes	12
4.1 Vector	12
4.2 Matrix	12
4.3 Matrix44	13
4.4 Matrix53	14
5 Other classes	15
5.1 Stream	15
5.2 FileStream , CompressedFileStream	15
II The Smalltalk interpreted scripting language	16
6 Introduction to Smalltalk	16

7	Language overview	16
7.1	Objects and messages	16
7.2	Variables	17
7.3	Arrays, sets and dictionaries	17
7.4	Control structures, loops and blocks	18
7.5	Strings	18
7.6	Numbers	19
7.7	Input/Output	19
7.8	Program header and method declaration	19
8	Illustrative examples	20
8.1	Navigating through the structure's hierarchy	20
8.2	Doing calculations	22
III	The GNUstep Foundation	25
9	Classes	25
9.1	NSString	25
9.2	NSNumber	25
9.3	NSArray, NSMutableArray	25
9.4	NSSet, NSMutableSet	26
9.5	NSDictionary, NSMutableDictionary	26
9.6	NSEnumerator	26
9.7	Transcript	26

MolTalk is programmed in Objective-C and comes in two different forms:

- *libmoltalk* (see I), a library, which can deal with structural information as stored in PDB files and maps this information to object-space.
- *MolTalk* (see II), the Smalltalk interpreter used as a front-end to *libmoltalk*.

The job MolTalk does, is in short:

- provides facility to easily load a PDB structure into memory
- creates objects to represent structure/chain/residue/atom hierarchy of PDB formatted files
- makes the correct associations between these objects
- provides user with a friendly interface to access all objects through object messaging

For rapid software development, users might start working with the scripting language as implemented in the *MolTalk* front-end. This scripting language has some advantages over compiled code: you do not need to know about C and Objective-C programming, dealing with GCC and the Smalltalk language is learnt very fast. Using one of the provided examples, as a starting point, it is easy to write his/her own script and launch the interpreter on it. Execution speed is acceptable for most applications.

For demanding applications where execution speed is critical, there is the possibility to program code directly in Objective-C and link with *libmoltalk*. With reasonable effort it is possible to port scripts in Smalltalk to Objective-C and compile them.

The rich set of classes and their methods makes *MolTalk* a key component in Structural Bioinformatics software development. MolTalk comes with the free GNU public licence allowing everybody to extend the library and adapt to his needs. Further, the licence allows the library to be distributed by everyone.

Part I

MolTalk Library

In this part you will learn about the basics of the MolTalk library. Classes are marked **Basic** whether their usage is easy and encouraged for the novice user, or, marked **Xtra** if they require more attention.

1 Requisites

To be able to compile Objective-C code and link with libmoltalk, you should have installed all the necessary tools on your system prior to continue.

GNU C compiler GCC: <http://gcc.gnu.org>
the current version 3.3.1 is recommended.

GNUstep: <http://www.gnustep.org>
the current version 1.8.0 is recommended. Install gnustep-make, then gnustep-base.

StepTalk: <http://steptalk.host.sk>
use the version provided by the CVS repository at <http://savannah.gnu.org/projects/gnustep>.
Check out the gnustep/dev-libs/StepTalk module.

For GNUstep base and make and StepTalk we provide copies of the source code from the MolTalk server
<http://www.moltalk.org>.

2 Class diagram

The class diagram in figure 1 shows the dependencies between classes in MolTalk. To see all the classes' attributes and methods, browse the class descriptions in the following chapters.

Xtra Creation of Structure, Chain, Residue:

StructureFactory (3.1), ChainFactory (3.3), ResidueFactory (3.5)

Basic Access to PDB file entity hierarchy:

Structure (3.2), Chain (3.4), Residue (3.6), Atom (3.7)

Basic PairwiseStrxAlignmnet (3.9), Selection (3.10)

Xtra Mathematics:

Matrix (4.2), Matrix44 (4.3), Matri53 (4.4)

Vector (4.1), Coordinates (3.8)

Xtra File access:

Stream (5.1), FileStream (5.2), CompressedFileStream (5.2)

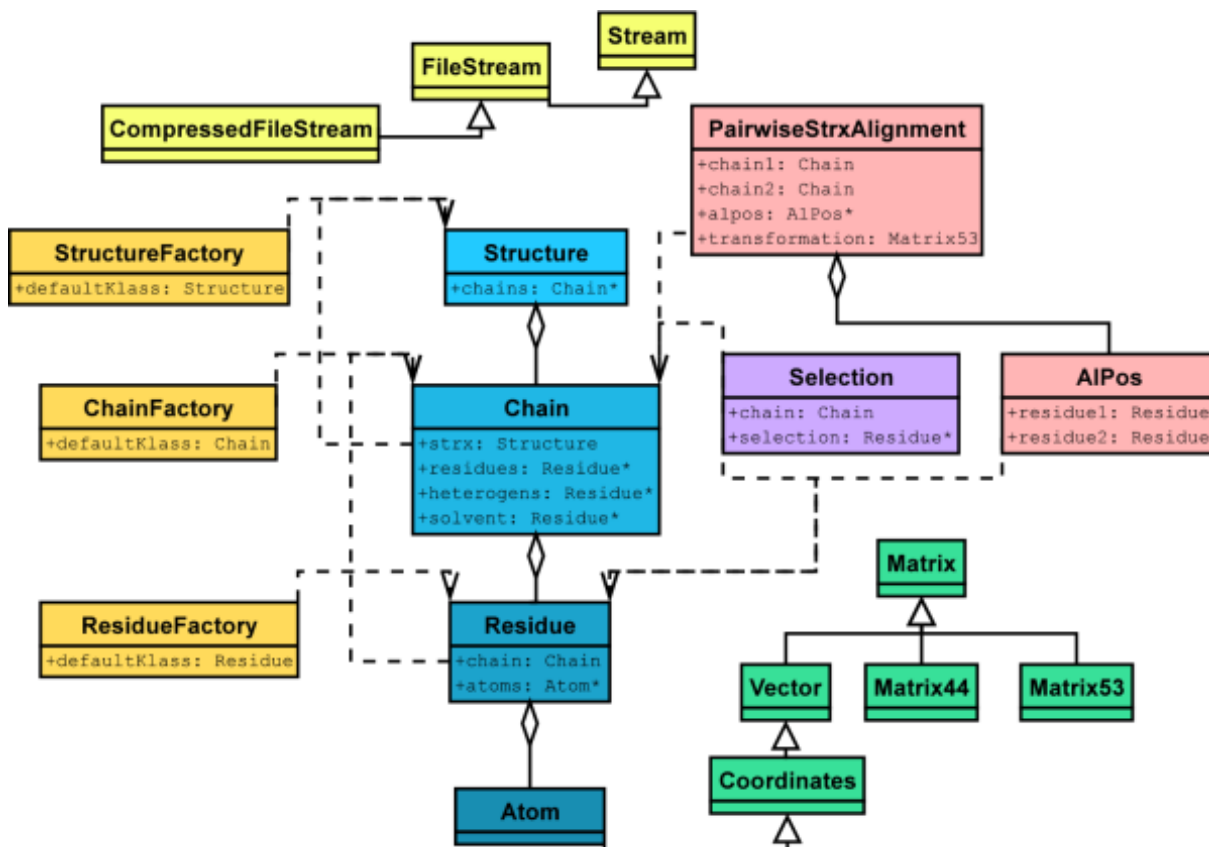


Figure 1: Class diagram showing the dependencies of the classes in MolTalk. Dashed lines indicate dependencies among classes. A triangle and line indicates inheritance between parent class and subclasses. A rhomb and line indicates that the second class is part of the first one.

3 Structural classes

3.1 StructureFactory

StructureFactory is used to create instances of class Structure (3.2). The environment variable PDBDIR must point to the local filesystem path of the PDB directory hierarchy as managed by PDBChainSaw or another mirroring tool.

Methods:

<code>+(id)newStructure</code>	This creates an empty structure
<code>+(id)newStructureFromPDBDirectory: (NSString*)code</code>	Basic Read structure from directory (note that the environment variable PDBDIR must be defined)
<code>+(id)newStructureFromPDBDirectory: (NSString*)code options: (long)opts</code>	Xtra also pass options <i>opts</i>
<code>+(id)newStructureFromPDBFile: (NSString*)fn</code>	Basic Read structure from the file system
<code>+(id)newStructureFromPDBFile: (NSString*)fn options: (long)opts</code>	Xtra also pass options <i>opts</i>

where *opts* is the sum of the options you want to pass to the parser:

PDBPARSER_ALL_NMRMODELS	1
PDBPARSER_IGNORE_SIDECHAINS	2
PDBPARSER_IGNORE_HETEROATOMS	4
PDBPARSER_IGNORE_SOLVENT	8
PDBPARSER_IGNORE_COMPOUND	16
PDBPARSER_IGNORE_SOURCE	32
PDBPARSER_IGNORE_KEYWORDS	64
PDBPARSER_IGNORE_EXPDTA	128
PDBPARSER_IGNORE_REMARK	256
PDBPARSER_IGNORE_REVDAT	512

- (ExperimentType) expdata	Basic returns the type of experimental method Structure_XRay = 100 Structure_NMR = 101 Structure_TheoreticalModel = 102 Structure_Other = 103 Structure_Unknown = 104
- (float) resolution	Basic resolution as in REMARK 2 lines
- (NSArray*) keywords	Basic returns an array of keywords as parsed from KEYWDS lines
- (NSString*) hetnameForKey: (NSString*) key	Basic HETNAME lines indicate long names for residue names. This method returns the long name for the key of a residue (see 3.6).
- (void) writePDBToStream: (Stream*) fn	Basic writes out the complete structure to a stream (see 5.1) in PDB format.
- (NSEnumerator*) allChains	Basic returns an enumerator over all chains
- (Chain*) getChain: (NSNumber*) p_chain	Basic returns the chain (see 3.4) for a given code
- (void) removeChain: (Chain*) p_chain	Basic remove a chain (see 3.4) from this structure

```

strx := StructureFactory newStructureFromPDBDirectory: '1GUA'.
chain := strx getChain: 66.                " access chain B = code 66. "
strx removeChain: chain.                  " remove the chain. "
stream := CompressedFileStream streamToFile: 'test0.pdb.gz'.    " create the stream we will write
to. "
strx writePDBToStream: stream.
stream close.                             " close this stream and force it to be written to disk. "

```

3.3 ChainFactory

Objects of type Chain are created by ChainFactory.

Methods:

+ (id) newChainWithCode: (char) code	Basic create a new Chain (see 3.4) with the given code
--	---

3.4 Chain

Class Chain holds three lists of residues (see 3.6): amino and nucleic acid residues (corresponding to ATOM entries in PDB files), heterogeneous groups (corresponding to HETATM entries in PDB files) and solvent residues (named HETATM residues with SOL, DIS, HOH).

The PDB file format indicates, that modified residues as stated in MODRES entries, have to be within the sequence of ATOM entries but marked as HETATM. We do respect this format decision on loading and writing PDB formatted files but treat this modified residues as normal residues in the chain's list of residues and mark them as being modified, which can be queried by the method *isModified* on the residue (see 3.6). Thus querying a chain's list of residues (not heterogens) returns also the modified residues.

Methods:

* naming	
- (char) code	Basic returns code of this chain
- (NSNumber*) numberCode	Basic returns the chain's code as a number
- (NSString*) name	Basic returns the code of this chain as a string
- (NSString*) description	Basic returns a textual description of this chain
- (NSString*) fullPDBCode	Basic concatenation of the structure's code and the chain's code
* read-only access	
- (NSString*) compound	Basic corresponding to COMPND lines in PDB files
- (NSString*) eccode	Basic corresponding to COMPND lines in PDB files with EC: entry
- (NSString*) source	Basic corresponding to SOURCE lines in PDB files

<code>-(Structure*)structure</code>	Basic reference to parent structure (see 3.2)
<code>-(id)transformBy: (Matrix53*)m</code>	Xtra transform all residues/atoms in this chain by the given matrix
* enumerator over residues, heterogens, solvent, respectively	
<code>-(NSEnumerator*)allResidues</code>	Basic returns an enumerator over all residues
<code>-(NSEnumerator*)allHeterogens</code>	Basic returns an enumerator over all heterogeneous residues
<code>-(NSEnumerator*)allSolvent</code>	Basic returns an enumerator over all solvent residue
* count of residues, heterogens, solvent, respectively	
<code>-(int)countResidues</code>	Basic returns the number of residues (amino acids and nucleic acids)
<code>-(int)countHeterogens</code>	Basic returns the number of heterogeneous residues
<code>-(int)countSolvent</code>	Basic returns the number of solvent residues
<code>-(int)countStandardAminoAcids</code>	Basic returns the number of standard (20 types) amino acids
* access a residue, heterogen, solvent, respectively, for the given identifying number	
<code>-(Residue*)getResidue: (NSString*)nr</code>	Basic search residue for its id
<code>-(Residue*)getHeterogen: (NSString*)nr</code>	Basic search heterogeneous residue for its id
<code>-(Residue*)getSolvent: (NSString*)nr</code>	Basic search solvent residue for its id
* add a residue, heterogen, solvent, respectively, to this chain	
<code>-(id)addResidue: (Residue*)res</code>	Basic add a new residue (see 3.6) to this chain
<code>-(id)addHeterogen: (Residue*)het</code>	add a new heterogen (see 3.6) to this chain
<code>-(id)addSolvent: (Residue*)sol</code>	Basic add a new solvent (see 3.6) to this chain
<code>-(void)removeResidue: (Residue*)p_res</code>	Basic remove a residue
<code>-(NSString*)get3DSequence</code>	Basic derive amino acid sequence from connected residues. Non-standard amino acids have single letter code 'X'
<code>-(NSString*)getSequence</code>	Basic derive amino acid sequence as above, but take into account chain breaks and fill missing residues with 'X'
<code>-(NSString*)getSEQRES</code>	Basic returns the sequence of amino acids as parsed from SEQRES lines.
<code>-(void)prepareResidueHash: (float)gridsize</code>	Xtra compute geometric hash of all residues. Recompute the geometric hash whenever the coordinates change (e.g. by the method <code>-(void)transformBy:</code>). The parameter <code>gridsize</code> is used to compute the number of bits, which encode the co-ordinates. Residues with close co-ordinates are stored in the same bin and can be found with <code>-findResiduesCloseTo:</code> .
<code>-(NSArray*)findResiduesCloseTo: (Coordinates*)p_coords</code>	Xtra find residues in this chain, which are close to the given Coordinates (see 3.8). The method <code>-(void)prepareResidueHash</code> must be called first.

3.5 ResidueFactory

Methods:

<code>+(id)newResidueWithNumber: (int)resnr name: (char*)name</code>	Basic create a new residue (see 3.6) with number and name
<code>+(id)newResidueWithNumber: (int)resnr subcode: (char)icode name: (char*)name</code>	Basic create a new residue (see 3.6) with number and insertion/deletion code, plus its name.

3.6 Residue

Class Residues represents a group of atoms (see 3.7) forming a nucleic acid residue, an amino acid residue or even a heterogeneous group (including solvent).

Methods:

```

* read-only access
- (Chain*)chain

- (NSString*)description
- (NSString*)name
- (NSString*)modname

- (NSString*)moddescription

- (NSString*)oneLetterCode

- (NSNumber*)number
- (char)subcode

- (NSString*)key

- (int)sequenceNumber

* tests
- (BOOL)isStandardAminoAcid

- (BOOL)isModified

- (BOOL)isNucleicAcid
- (BOOL)haveAtomsPresent

- (double)distanceCAto: (Residue*)r2

* atoms
- (id)addAtom: (Atom*)atom
- (NSEnumerator*)allAtoms
- (Atom*)getAtomWithName:
  (NSString*)name
- (Atom*)getAtomWithInt: (unsigned
  int)number
- (Atom*)getAtomWithNumber:
  (NSNumber*)number
- (Atom*)getCA

* manipulation
- (id)transformBy: (Matrix53*)m

```

Basic access the parent Chain (see 3.4) of which we are part

Basic returns a textual description of this residue

Basic returns the name

Xtra returns the name of the standard amino acid as the base of this modified residue. This is only used in case the residue is modified as indicated by MODRES entries in the PDB file.

Xtra returns the description of the modification as indicated by a MODRES entry in the PDB file

Basic Translate the name of this residue to the one letter amino acid code. If it is not known this returns '@'X'.

Basic returns the number

Basic This field is only used for distinguishing different residues of the same number. The PDB describes this as the insertion/deletion code.

Basic returns a concatenation of number and subcode, as this uniquely identifies this residue.

Basic returns the residue's sequential number in the chains sequence. This number is set upon a call to Chain_getSequence or Chain_get3DSequence (see 3.4)

Basic true if this residue is one of the 20 standard amino acids

Xtra true if this residue is modified as indicated by a MODRES entry in the PDB file

Basic true if this residue is a nucleic acid

Basic true if the connectivity of this residue's atoms is known to be correct

Basic calculate and return the distance between the C_{α} atoms of the two residues. If the distance cannot be calculated due to missing atoms, the method returns -1.0.

Basic include new atom in the atom set of this residue

Basic return an enumerator over all atoms

Basic return atom with the given name

Basic return atom with the given integer

Basic return atom with the given number

Basic special case of getAtomWithName:@'CA'

Basic transform all atoms of this residue

Objects of type Residue are created by ResidueFactory (see 3.5).

```

res1 := currchain getResidue:'108'.           " find residue. "
res2 := currchain getResidue:'47'.           " find another residue. "
distance := res1 distanceCAto: res2.         " calculate CA distance "

```

3.7 Atom

Objects of class Atom represent ATOM and HETATM lines in PDB files.

NEW: every atom can have a partial charge and is of a defined chemical type (element). This information is parsed from and written to PDB files.

Class Atom inherits from class Coordinates (see 3.8) and implements the additional methods:

```

* read-only access
-(NSString*)name
-(NSNumber*)number
-(double)temperature
-(int)element

-(NSString*)elementName
-(int)charge
* bonding
-(NSEnumerator*)allBondedAtoms
-(void)bondTo: (Atom*)atm2
-(void)dropAllBonds
-(void)dropBondTo:(Atom*)atm2
*setters
-(id)setNumber: (int)serial
-(id)setCharge: (int)chrg
-(id)setElement: (int)elid
-(id)setElementWithName:
(char*)elname

+(Atom*)atomWithNumber: (unsigned
int)num name: (char*)nm X:(double)x
Y:(double)y Z:(double)z B:(double)b

```

Basic returns name of atom
Basic returns serial number of atom
Basic returns temperature factor of atom
Basic returns index number of chemical element of this atom:
ELEMENT_ID_H = 1
ELEMENT_ID_C = 6
ELEMENT_ID_N = 7
ELEMENT_ID_O = 8
...
ELEMENT_ID_Unknown = -1
Just to name a few.
Basic returns string of element name
Basic returns partial charge on atom (normally 0)
Basic returns an enumerator over all bonded atoms
Basic add a bond from this atom to the given atm2
Basic remove all bonds
Basic remove the bond to atm2
Xtra update atom serial number
Xtra update partial charge on atom
Xtra set atom to be of chemical type: elid
Xtra derive chemical type from atom name
Basic creation of a new atom with the given information

3.8 Coordinates

Class Coordinates inherits from class Vector (see 4.1). It represents the special case of a three dimensional vector, though it is of four dimensions and can be multiplied with a 4x4 matrix (see 4.3). In addition to the methods inherited from class Vector it implements:

```

* calculate distance
-(double)distanceTo:
(Coordinates*)c2
-(double)distanceToLineFrom:
(Coordinates*)v2 to:
(Coordinates*)v3
-(double)x
-(double)y
-(double)z
-(id)setX:(double)newx Y:(double)newy
Z:(double)newz
-(id)rotateBy:(Matrix44*)m
-(id)transformBy: (Matrix53*)m
* creation

```

Basic calculates the distance between two coordinates
Basic calculates the distance from this point to the line given by (v2,v3)
Basic returns X coordinate
Basic returns Y coordinate
Basic returns Z coordinate
Basic sets new coordinates
Basic apply Matrix44 (see 4.3) to this coordinates
Basic apply Matrix53 (see 4.4) to this coordinates

```

+(Coordinates*) coordsFromCoordinates: Basic make a copy of another coordinates
(Coordinates*) p_coords
+(Coordinates*) coordsWithX: (double)x Basic create new with explicit coordinates
Y: (double)y Z: (double)z
+(Coordinates*) origin Basic create new at (0,0,0)

```

3.9 PairwiseStrxAlignment

Structural alignments between two chains (see 3.4) can be computed with class PairwiseStrxAlignment.

Methods:

```

* read-only access
-(NSString*) description Basic returns textual representation
-(Chain*) chain1 Basic returns first chain (see 3.4)
-(Chain*) chain2 Basic returns second chain (see 3.4)
-(NSArray*) alignmentPositions Xtra returns array of alignment positions as an array
(see 9.3)
-(Matrix53*) getTransformation Basic returns calculated transformation (see 4.4) on
chain2

* operations
-(void) deriveStructuralAlignment Basic compute transformation based on superimposed
chains. This method uses dynamic programming with a
scoring system based on pairwise distances.
Xtra recompute transformation from selecting residue
pairs, which have at most 3.5 A distance
-(void) optimize Basic calculate RMSD of structural alignment
Basic count alignment positions in structural alignment
Basic count aligned pairs only
Basic count aligned pairs with distance below given
distance
-(double) calculateRMSD
-(int) countPairs
-(int) countUngappedPairs
-(int) countPairsMaxDistance:
(double) dist
* input/output
-(void) fromStreamAsTCoffee: Xtra remake structural alignment based on the pairwise
(Stream*) stream information in the T_Coffee read from stream
-(void) toStreamAsTCoffee: Xtra write structural alignment in T_Coffee library
(Stream*) stream format to the given stream, explicitly name chain
name1: (NSString*) name1 sequences
name2: (NSString*) name2
* creation
+(PairwiseStrxAlignment*) alignmentBetween: Basic create a new instance working on chain1 and
(Chain*) chain1 chain2
andChain: (Chain*) chain2

```

```

strxal := PairwiseStrxAlignment alignmentBetween:chain1 and chain2. " create object. "
strxal deriveStructuralAlignment. " from the given superimposition, derive the structural alignment
(alignment positions). "

```

3.10 Selection

Class Selection holds a reference to a single chain (see 3.4) and a list of a subset of its residues (see 3.6) that are selected.

Methods:

```

* access
-(NSString*) description Basic returns textual representation

```

```

-(unsigned long)count
-(NSEnumerator*)selectedResidues

* operations
-(id)addResidue: (Residue*)r
-(id)removeResidue: (Residue*)r
-(id)union:(Selection*)sel2
-(id)difference:(Selection*)sel2
* structural alignment of two selections
-(Matrix53*)alignTo:
(Selection*)sel2
* creation
+(Selection*)selectionWithChain:
(Chain*)c

```

Basic count the number of residues in this selection
Basic returns enumerator (see 9.6) over selected residues

Basic include residue in selection

Basic exclude residue from selection

Basic add all selected residues from *sel2* to this one

Basic exclude all selected residues in *sel2* from this one

Xtra structurally align this selection to the other one and return the resulting transformation

Basic creates a new selection acting on chain *c*

4 Mathematical classes

4.1 Vector

Class Vector has subclass: Coordinates (see 3.8).

Class Vector inherits from class Matrix (see 4.2) and implements the additional methods:

* access	
- (NSString*)description	Basic returns textual description: [dim0,dim1,...,dimn-1]
- (NSString*)toString	Basic same as -(NSString*)description
- (int)dimension	Basic number of dimensions
- (double)atDim: (int)dim	Basic returns value at given dimension
- (id)atDim: (int)dim value: (double)v	Basic set value at given dimension
* operations	
- (double)length	Basic returns length of the vector
- (double)eukclideanDistanceTo: (Vector*)v2	Basic distance between two vectors
- (id)differenceTo: (Vector*)v2	Basic returns difference vector of the two
- (id)add: (Vector*)v2	Basic add another vector to this one
- (id)scaleByScalar: (double)scalar	Basic multiply all coordinates by the scalar
- (id)normalize	Basic scale by 1/length so vector becomes of length 1.0
- (double)angleBetween: (Vector*)v2	Basic calculates angle between two vectors
- (double)scalarProductBy: (Vector*)v2	Basic $v1 * v2$
- (id)vectorProductBy: (Vector*)v2	Basic $v1 \times v2$
- (double)mixedProductBy: (Vector*)v2 and: (Vector*)v3	Basic $v1 * (v2 \times v3)$
* creation	
- (id)setDimensions: (int)dim	Xtra set dimensions to new length, destroys content
+ (Vector*)vectorWithDimensions: (int)dim	Basic create a new vector with given dimensions

4.2 Matrix

Subclasses of class Matrix are: Matrix44 (see 4.3), Matrix53 (see 4.4) and Vector (see 4.1).

Methods:

* read-only access	
- (NSString*)description	Basic returns textual representation of this matrix in the form: [[first row]...[last row]]
- (NSString*)toString	Basic same as -(NSString*)description
- (int)cols	Basic returns number of rows
- (int)rows	Basic returns number of cols
- (BOOL)isTransposed	Basic true if this matrix is transposed (rows and cols exchanged)
* getter	
- (double)atRow: (int)row col: (int)col	Basic returns value at <i>row/col</i>
- (void)linearizeTo: (double*)mat maxElements: (int)count	Basic puts all values in row first order into the array of doubles <i>mat</i> , at most <i>count</i> elements.
- (id)matrixOfColumn: (int)col	Basic returns a $n \times 1$ matrix of all values in column <i>col</i> (start counting at 0)
* setter	
- (id)atRow: (int)row col: (int)col value: (double)v	Basic sets value <i>v</i> at <i>row/col</i>
- (id)atRow: (int)row col: (int)col add: (double)v	Basic add value <i>v</i> to value at element <i>row/col</i> (in place!)
- (id)atRow: (int)row col: (int)col subtract: (double)v	Basic subtract value <i>v</i> from value at element <i>row/col</i> (in place!)

```

-(id)atRow:(int)row col:(int)col
multiplyBy:(double)v
-(id)atCol:(int)col row:(int)row
divideBy:(double)v
* matrix operations
-(id)transpose
-(double)sum
-(id)square
-(id)addScalar: (double)scal
-(id)subtractScalar: (double)scal
-(id)multiplyByScalar: (double)scal
-(id)divideByScalar: (double)scal
-(id)madd: (Matrix*)m2

-(id)msubtract: (Matrix*)m2

-(id)mmultiply: (Matrix*)m2

-(id)x: (Matrix*)m2

-(Matrix*)centerOfMass

-(Matrix*)jacobianDiagonalizeWithMaxError:(double)error

* creation
+(Matrix*)matrixWithRows:(int)row
cols:(int)col
-(id)initFromString:(NSString*)str
-(id)setRows:(int)row cols:(int)col

```

Basic multiply element at *row/col* with value *v* (in place!)

Basic divide element at *row/col* with value *v* (in place!)

Basic transposes this matrix

Basic calculates the sum of all elements

Basic squares every element (in place!)

Basic adds a scalar to every element (in place!)

Basic subtracts scalar from every element (in place!)

Basic multiplies every element by scalar (in place!)

Basic divides every element by scalar (in place!)

Basic returns the resulting matrix from adding matrix *m2* to this one

Basic returns the resulting matrix from subtracting matrix *m2* from this one

Basic pairwise multiplies elements in the matrix and returns resulting matrix

Basic returns the resulting matrix of matrix multiplication of this with *m2*

Basic returns a 1xn matrix with the weighted sum of all values per columns

Basic diagonalises the matrix to given error, and returns matrix of eigenvectors in columns, eigenvalue in first row of column

Basic create a new matrix with rows/cols allocated

Basic reinitialise the matrix from parsing the string

Xtra resets the number of rows/cols, this destroys the content of the matrix!

4.3 Matrix44

Class Matrix44 inherits from class Matrix (see 4.2). It represents a rotation and translation to be performed on a vector.

```

r r r x
r r r y
r r r z
0 0 0 1

```

The rotation part is the 3x3 matrix with elements 'r'. The translation is the third column denoted as $(xyz1)^T$.

Methods:

```

* manipulation
-(Matrix44*)invert
-(Matrix44*)xIP: (Matrix44*)m2

-(Matrix44*)chainWith: (Matrix44*)m2

* creation
+(Matrix44*)matrixFromString:
(NSString*)strx
+(id)matrixIdentity

```

Xtra invert the 4x4 rotation matrix

Xtra matrix multiplication with matrix *m2* (in place!):
 $M' = M * M2$

Xtra matrix multiplication with matrix *m2* (in place!):
 $M' = M2 * M$

Basic creates a new 4x4 matrix, initialised by values from parsed string

Basic creates a new 4x4 matrix with values:

```

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

```

4.4 Matrix53

Class Matrix53 inherits from class Matrix (see 4.2).

This matrix comprises a Matrix44 for rotation and translation, plus a translation vector to be applied before rotation (usually the translation back to the origin).

r r r

r r r

r r r

o o o

t t t

$$\mathbf{v}' = R * (\mathbf{v} - \mathbf{o}) + \mathbf{t}$$

Methods:

-(Matrix53*)invert

Xtra inverts the rotation part of the 5x3 matrix and the origin part

* getter

-(Coordinates*)getOrigin

Basic returns the coordinates (see 3.8) of the origin part

-(Matrix44*)getRotation

Basic returns the rotation part (see 4.3)

-(Matrix44*)getTranslation

Basic returns the translation part as a 4x4 matrix (see 4.3) (useful for subsequent matrix multiplication):

```
1 0 0 tx
0 1 0 ty
0 0 1 tz
0 0 0 1
```

* creation

+(Matrix53*)matrixFromString:(NSString*)

Basic creates a new 5x3 matrix with values from parsing string *str*

+(id)matrixIdentity

Basic creates a new 5x3 matrix with values:

```
1 0 0
0 1 0
0 0 1
0 0 0
0 0 0
```

5 Other classes

5.1 Stream

Class Stream is an abstract class and is not intended to be instantiated but to serve as a root class to all sorts of streams, like implemented in classes FileStream (see 5.2) and CompressedFileStream (see 5.2). This way we make sure that everything that accepts a stream for input or output accesses it through the methods of class Stream whether it is a FileStream or a CompressedStream (and in the future maybe a HttpStream).

Methods:

* test	
-(BOOL)ok	Basic returns true if the stream is still valid
* operation	
-(void)close	Basic close the stream and invalidate it
* write access	
-(void)writeCString: (const char*)string	Basic write C string to the stream
-(void)writeData: (NSData*)data	Basic write data to the stream
-(void)writeString: (NSString*)string	Basic write string to the stream
* read access	
-(NSData*)readLength: (unsigned int)len	Basic read data with maximum length from stream
-(NSData*)readLineLength: (unsigned int)len	Basic read data line up to the end of line or maximum length
-(NSString*)readStringLineLength: (unsigned int)len	Basic read string up to the end of line or maximum length

5.2 FileStream , CompressedFileStream

Class FileStream and CompressedFileStream are subclasses of class Stream (see 5.1). CompressedFileStream can read from and write to a compressed file using the tool *gzip*, which must be installed and accessible on the system. In addition to the methods available from class Stream, these classes also implement the following methods:

* creation	
+(id)streamFromFile: (NSString*)path	Basic create stream to read from file
+(id)streamAppendToFile: (NSString*)path	Basic create stream to append to file
+(id)streamToFile: (NSString*)path	Basic create stream to write to file

Part II

The Smalltalk interpreted scripting language

In this part you will get a glance at the basic concept behind Smalltalk, its implementation in StepTalk and how you can make use of it with MolTalk.

The examples are implemented in the scripting language and can be executed by the interpreter *MolTalk*.

6 Introduction to Smalltalk

Smalltalk is the “most” object-oriented programming language in the universe. In its design there is no compromise between abstract objects and machine-dependent “types”. Everything is an object!

What is an object?

- Every object is unique, thus we can access an object by its unique identification. (Think of something like car plates or telephone numbers.)
- Every object is of a specified class. A class is just the meta information that defines objects. A class can actually create new objects of its type.
- Objects do respond to messages, which may take parameters or not.
- Further, we need to consider an object’s lifetime, that is, the duration of the existence of the object, starting with its creation, lasting till its destruction. You control it!

Others? There are none, thus everything is an object. A Smalltalk program lives in an environment full of objects. It does only see objects and sends messages to them following your programming code, then, waits for the message to return and continues.

Why objects?

You may have wondered why there is a need for such things like object-oriented programming. Considering MolTalk as an application of object-oriented programming it jumps right at your eyes: you get access to information (macromolecular structures) without caring about their intestines, you want to know the objects’ ids and then access them. So easy!

7 Language overview

MolTalk is an interpreter of the Smalltalk language, based on StepTalk. It takes an input file, which is the code you wrote. Starting at the very beginning it finds its way through that code and compiles it into an internally hidden byte-code format about which we don’t care. This code is a representation of all the commands and arguments, and so on, of your script file. MolTalk then sets out to interpret this internal structure byte by byte.

Comments are written between “ and “ as many lines long as you wish. Use this feature to document your scripts. They will not harm the speed of interpretation.

At the start of the interpreter, an array (see 9.3) named ARGV is built, which contains all the parameters that have been passed on the command line after the script name. Then, the method named “main” is executed by the interpreter.

7.1 Objects and messages

Smalltalk is about communication between objects. This manifests in sending messages to objects. Messages are synchronous, meaning, that the program does not continue unless the one message sent out has returned.

```
anObject messageWithParameter: param.           " anObject is an object and receives the message
messageWithParameter: with a parameter named param, which itself is an object "
```

The above code is a complete command: it states the receiver (`anObject`), the message to pass (`messageWithParameter:`) and the parameter to pass (`param`). IMPORTANT: A command ends with a dot `'.'` to indicate the end of the message list.

But we can also pass messages, which do not take any parameter:

```
anObject justDoSomething.           " anObject is the receiver of the message justDoSomething. There is no
parameter being passed "
```

We can force the order of the evaluation of the parameters by grouping the message passing with brackets:

```
anObject msg1: ((anotherObject msg2:param) doSomething).           " the innermost command is
evaluated, the message doSomething sent to it, the resulting evaluation passed to object anObject via the
message msg1: "
```

By default, the order of evaluation is left to right.

7.2 Variables

In the above examples “`anObject`” is rather a named variable which contains a reference to an object. Through such variables we communicate with objects.

Variables are initialised with the id of (or a reference to) an object:

```
myAge := 17.                       " implicit creation of a number object "
anObject := NSObject new.          " explicitly create an object by calling new on its class "
myString := 'this is indeed a string'. " implicit creation of a string object "
```

The evaluation of the right part (as complex it might be) is a reference to an object in any case. This evaluation is stored in the named variable as stated on the left part. The assignment operator “`:=`” is special and distinguishes from “`=`”, which is an evaluation operator testing for equality/identity.

7.3 Arrays, sets and dictionaries

As objects are accessible by their id only, we have to make sure we remember it. If we forget about this id, we cannot send messages to the object anymore. The object would just be lost, albeit still reside alive in the computer’s memory. In case we have to deal with only a limited number of objects, we can have them remembered in variables. But if we start having dozens, hundreds or even thousands of objects, this approach becomes infeasible.

A simple method of storing object ids are lists or arrays. You put your objects into the list, then only remember the list’s id. By accessing the list you will get to your objects again.

Several types of lists will help you to put your objects in order: `NSArray`(9.3), `NSSet`(9.4) and `NSDictionary`(9.5). These classes are defined in the GNUstep Foundation and accessible by Smalltalk as well as compiled Objective-C programs.

A short form of array construction in SmallTalk is: `#(element1,element2,element3)`. Use this only if you will forever keep your code in SmallTalk, as this form does not exist in Objective-C.

Example:

```
fixedArray := #(1,3,4,5,99,-3,1.2).           " creates an array of some numbers "
myArray := NSMutableArray new.                " create an array of class NSMutableArray, which can be altered "
myArray addObject: myFonNumber.              " add an object to the array "
```

7.4 Control structures, loops and blocks

Blocks of program code can be accessed like objects. They can be passed as parameters to messages, or, receive messages with optional parameters and thus become executed.

A simple example for conditional block execution:

```
(3 > 2) ifTrue: [ res := 'YES' ] ifFalse: [ res := 'NO' ].
```

The first part “(3 > 2)” is evaluated to a boolean, which then receives the message “ifTrue:ifFalse:” with the two blocks passed as parameters. An object of class boolean is executing the first block if itself evaluates to true, otherwise it executes the second block. The blocks do not receive any values.

We define blocks by putting the code between “[” and “]”. At the beginning of the block we state a list of the receiving, local variables for the parameters, each prefixed with a colon (“:arg1”). Local variable declaration has to be separated from the block’s code by a “|”.

Example: Entries in NSArray(9.3) lists are accessed by their indexes.

```
theElement := myArray objectAtIndex: 2.           " access third element, start counting at 0 (zero) "
0 to: (myArray count) - 1 do: [ :idx |           " count from 0 to the length -1, executes block on every
number, which is received in idx "
    theElement := myArray objectAtIndex: idx.    " get element at index given by the local variable idx
"
].
```

NSSet (9.4) lists are accessed by qualitative means.

Example:

```
(theSet containsObject: myObj) ifTrue: [ Transcript showLine: 'yeah, have it there!' ].
```

Or, by enumerating all the elements it contains.

Example:

```
(theSet allObjects) do: [ :elem | Transcript showLine: elem description ].
```

Same holds for NSDictionary (9.5). We can access all the values it contains, all the keys, or, a single object for a key:

```
(theDictionary allKeys) do: [ :key | Transcript showLine: key description ].
(theDictionary allValues) do: [ :value | Transcript showLine: value description ].
aValue := theDictionary objectForKey:'I am a key'.           " retrieve value or nil "
(aValue notNil) ifTrue: [ res := 'YES' ] ifFalse: [ res := 'NO' ]. " test for valid value "
```

7.5 Strings

Text is stored in NSString (9.1).

Creation of strings is very easy:

Example:

```
myStr := 'this is an example of a string'.           " implicit creation by SmallTalk "
aNumber := 333333.888877.
anotherStr := aNumber stringValue.                 " conversion to a string "
myStr := NSString stringWithCharArray: #( 65, 66, 67, 68, 69 ). " returns string 'ABCDE' "
```

They can be concatenated:

Example:

```
newStr := oldStr stringByAppendingString: ' yes yes yes'.
newStr := oldStr , ' and even more.'. " short form of string concatenation "
```

7.6 Numbers

In SmallTalk numbers are objects of class NSNumber (9.2), but nevertheless, we can do calculation with them. The common operators acting on them are available.

Example:

```
nOne := 1. " implicit creation "
nTwo := 3.75 * 28.
nThree := nOne - nTwo.
```

Note: The operators must be surrounded with spaces. The interpreter will not correctly recognise them, otherwise.

7.7 Input/Output

The user's console is represented as class/object Transcript (see 9.7) and available for writing output.

Example:

```
Transcript showLine: ('Here comes an object: ', myObject description).
```

This prints a line on the Transcript. The string being printed is a concatenation of the fixed string 'Here ...' and the evaluation of the message description to the object myObject, which results in a string describing the object.

7.8 Program header and method declaration

A program needs the following:

- start with the tag “[]”
- your code
- and end with “]”

At least the “main” method must be defined in your code. It will be called by the interpreter to run your script. Your code might actually implement sub-methods as well. A method declaration looks like:

1. start with the tag “!” (required only for sub-method declaration)
2. give the name of the method (e.g. main)
3. possibly have a declaration of local variables: “[]”, a list of their names, separated by spaces, and “[]” again
4. next are the objects and their messages as your programming code
5. and last is the return statement: **^self** (or another value)

To call a sub-method you just created in your program, pass its name and the parameters (if there are some) to the object “self”.

Example:

```
[ |
main
  | myVariable |
myVariable := 333.87 .
self printValue: myVariable.          " call ourself with message printValue: and pass myVariable as
parameter "
!
printValue: val          " declaration of method printValue: which receives a paramter in the local variable val "
| localVar |
localVar := val description.          " call the message description on val "
Transcript showLine: ('value = ', localVar).
^self          " return our identification (could be any object) back to caller "
]
```

8 Illustrative examples

8.1 Navigating through the structure's hierarchy

Loading a structure and traversing the structure/chain/residue hierarchy, outputting some information.

```
" Copyright 2003 Alexander V. Diemand
This file is part of MolTalk.
MolTalk is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
MolTalk is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with MolTalk; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
"

[ |

" traverses the structure/chain/residue hierarchy
at the end it prints out the count of residues
with CA distances in the range from 3.5 to 4.1
grouped by tenth of Angstroms
"

main

  | arg strxcode |

strxcode := nil.
1 to: (ARGS count) - 1 do: [ :argc |
  arg := (ARGS objectAtIndex:(argc intValue)).
  (arg = '-strx') ifTrue: [ targc := argc + 1. strxcode := (ARGS
objectAtIndex:(targc)). ].
].
(strxcode isNil) ifTrue: [ self usage. ]
ifFalse: [ self continue:strxcode. ].
^self
```

```

!
usage

Transcript showLine:'Usage: MolTalk scripts/traverseHierarchy.st -strx [strxcode]'.
Transcript showLine:".
^self

!
continue:strxcode

| strx CAatm1 CAatm2 dist idx distr val |

strx := StructureFactory newStructureFromPDBDirectory: strxcode.          " Structure3.2 "
(strx notNil) ifTrue: [
    distr := NSMutableArray arrayWithArray: #(0 0 0 0 0 0).              " NSArray9.3 "
    Transcript showLine:( 'have structure:',strx pdbcode).
    Transcript showLine:( 'header: ',strx header).
    Transcript showLine:( 'title: ',strx title).
    Transcript showLine:( 'date: ',strx date description).
    (strx allChains allObjects) do: [ :chain |                            " Chain3.4 "
        Transcript showLine:( ' have chain:',chain description,'#',chain code stringValue).
        Transcript showLine:( ' E.C. code: ',chain eccode).
        Transcript showLine:( ' compound: ',chain compound).
        Transcript showLine:( ' source: ',chain source).
        Transcript showLine:( ' #aa: ',chain allResidues allObjects count stringValue).
        Transcript showLine:( ' #het:',chain allHeterogens allObjects count stringValue).
        Transcript showLine:( ' #sol:',chain allSolvent allObjects count stringValue).
        CAatm2 := nil.
        (chain allResidues allObjects) do: [ :residue |                  " Residue3.6 "
            CAatm1 := residue getCA.                                     " Atom3.7 "
            (CAatm2 notNil) ifTrue: [
                dist := CAatm1 distanceTo: CAatm2.
                idx := ((dist - 3.5) * 10) intValue.
                ((idx >= 0) and: (idx < 6)) ifTrue: [
                    val := distr objectAtIndex:index.
                    distr replaceObjectAtIndex:index withObject:(val + 1).
                ].
                (dist > 4.0) ifTrue: [
                    Transcript showLine:( 'CA-CA distance > 4.0 A between residues: ',presidue
description, ' and ',residue description, ' distance=',dist stringValue).
                ].
            ].
            " if we know the previous CA atom "
            residue := residue.          " remember this residue in next iteration as previous residue "
            CAatm2 := CAatm1.           " remember this CA atom "
        ].
        " for all residues "
        Transcript showLine:".
    ].
    " for all chains "
    " if strx "
]
iffalse: [ Transcript showLine:'I do not know this structure.'. ].

0 to: 5 do: [ :index |
    Transcript showLine:( 'count of residues with CA distance between ',
((35 + index) / 10.0) stringValue, ' and ',
((36 + index) / 10.0) stringValue, ' are: ',
(distr objectAtIndex:index) stringValue).
].
^self
]

```

8.2 Doing calculations

Loading a structure and outputting the ϕ/ψ angles of all residues.

```

" Copyright 2003 Alexander V. Diemand
This file is part of MolTalk.
MolTalk is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
MolTalk is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with MolTalk; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
"

[ |

" calculate Phi, Psi, and Chi angles of every residue in chain"

" plotting with gnuplot:
* extract the last part of the output and put it in file test.phipsi
* start gnuplot and enter: plot [-180:180] [-180:180] 'test.phipsi'
"

main

| arg strxcode chaincode |

strxcode := nil.
chaincode := 0.
1 to: (ARGS count) - 1 do: [ :argc |
    arg := (ARGS objectAtIndex:(argc intValue)).
    (arg = '-strx') ifTrue: [ targc := argc + 1. strxcode := (ARGS
objectAtIndex:(targc)). ].
    (arg = '-chain') ifTrue: [ targc := argc + 1. chaincode := (ARGS
objectAtIndex:(targc)) characterAtIndex:0. ].
    ].
    ((strxcode isNil) or: (chaincode < 10)) ifTrue: [ self usage. ]
    ifFalse: [ self continue:strxcode with:chaincode. ]
    ^self

!
usage

Transcript showLine:'Calculate Phi/Psi angles around backbone of all residues in
structural chain.'.
Transcript showLine:'Usage: MolTalk scripts/calc_phi_psi.st -strx [strxcode] -chain
[chaincode]'.
Transcript showLine:".

^self

!
continue:strxcode with:chaincode

```

```

| strx chain |

Transcript showLine: ('got:', strxcode). " Transcript9.7 "
strx := StructureFactory newStructureFromPDBDirectory: strxcode. " StructureFactory3.1
"
(strx notNil) ifTrue: [ " Structure3.2 "
    Transcript showLine: 'have structure'.
    chain := strx getChain: chaincode. " Chain3.4 "
    (chain notNil) ifTrue: [
        Transcript showLine: 'have chain'.

        self calculatePhiPsiForChain: chain. " do the calculation for all residues in chain "
    ]

    ifFalse: [ Transcript showLine: 'no such structure found.' ].
]

ifFalse: [ Transcript showLine: 'no such chain found.' ].

^self

!
calculatePhiPsiForChain:chain

| chain resphi respsi resp dist Phi Psi CAatom Catom C2atom Natom N2atom |

resp := nil.
resphi := NSMutableDictionary new. " see 9.5 "
respsi := NSMutableDictionary new.
(chain allResidues allObjects) do: [ :residue | " Chain:3.4 Residue:3.6 "
    (resp notNil) ifTrue: [
        atm1 := residue getCA. " Atom:3.7 "
        atm2 := resp getCA.
        dist := atm1 distanceTo: atm2.
        Transcript showLine:(' CA-CA distance: ',dist stringValue).
        (dist < 4.0) ifTrue: [
            "calculate Psi (around Ca-C)"

            CAatm := resp getCA.
            Natm := resp getAtomWithName:'N'.
            Catm := resp getAtomWithName:'C'.
            N2atm := residue getAtomWithName:'N'.
            (((CAatm notNil) and: (Catm notNil)) and: ((N2atm notNil) and: (Natm notNil)))
        ]
        ifTrue: [
            Psi := self calculateDihedralAngleBetween:Natm and:CAatm to:Catm and:N2atm.
            Transcript showLine: ('Residue: ',resp description, ' Psi=',Psi
stringValue).
            respsi setObject:Psi forKey:(resp description).
        ].
        " calculate Phi (around Ca-N) "

        CAatm := residue getCA.
        Catm := residue getAtomWithName:'C'.
        Natm := residue getAtomWithName:'N'.
        C2atm := resp getAtomWithName:'C'.
        (((CAatm notNil) and: (Catm notNil)) and: ((C2atm notNil) and: (Natm notNil)))
    ]
    ifTrue: [
        Phi := self calculateDihedralAngleBetween:C2atm and:Natm to:CAatm and:Catm.
        Transcript showLine: ('Residue: ',residue description, ' Phi=',Phi
stringValue).
        resphi setObject:Phi forKey:(residue description).
    ].
]

```

```

    ifFalse: [ Transcript showLine: 'CA-CA distance > 4.0 A, do not assume direct
bonding between consecutive amino acids in sequence.' ].
    ] " if we know the previous residue "
    ifFalse: [ Transcript showLine: 'previous residue not found.' ].
    resp := residue. " remember this residue in next iteration as previous residue "
    ]. " for all residues "
    Transcript showLine: 'done.'.
    Transcript showLine: '* * * * *'.
* * * * *.
(resphi allKeys) do: [:residue |
    Phi := resphi objectForKey: residue.
    Psi := respsi objectForKey: residue.
    (Psi notNil) ifTrue: [ " output only if Phi and Psi known "
        Transcript showLine: (Phi stringValue, ' ', Psi stringValue).
    ].
].
^self

!
calculateDihedralAngleBetween:P1 and:P2 to:P3 and:P4

" This function calculates the dihedral angle between two vectors
(each given by its starting and ending point)
"

| angle V21 V23 V32 V34 N1 N2 Sign Spat |

V21 := P1 differenceTo: P2. " points define vectors; see 4.1 "
V23 := P3 differenceTo: P2.
V32 := P2 differenceTo: P3.
V34 := P4 differenceTo: P3.
N1 := V21 vectorProductBy: V23.
N2 := V32 vectorProductBy: V34.
Spat := V23 mixedProductBy: N1 and: N2.
(Spat >= 0.0) ifTrue: [ Sign := -1.0 ] ifFalse: [ Sign := 1.0 ].
angle := (N1 angleBetween: N2) * Sign.
^angle
]

```

Part III

The GNUstep Foundation

The GNUstep project aims at providing a free implementation of the published interface of the NextStep development environment (Foundation). A rich set of classes helps software engineers to write programs in a portable and efficient code. Programs which are based on GNUstep Foundation classes can be easily ported to NextStep or even the new MacOSX.

In this section we will briefly sketch some of the most used classes. For full documentation please visit the GNUstep project at <http://www.gnustep.org>.

9 Classes

9.1 NSString

<code>+(id)stringWithFormat:,...</code>	creates a string following the format as defined in the first parameter. Further parameters are used to fill in the format. This method cannot be called from the MolTalk scripting language.
<code>+(id)stringFromArray:(NSArray*)astr</code>	addition to NSString to create a string from an array of character codes (instances of NSNumber9.2)
<code>-(float)floatValue</code>	return converted real numbers
<code>-(double)doubleValue</code>	
<code>-(int)intValue -(long)longValue</code>	return converted integer numbers
<code>-(NSString*)stringByAppendingString:(NSString*)str</code>	return a new string, which is the concatenation of the two
<code>-(NSString*)substringToIndex:(int)idx</code>	returns substring up to but not including index (start counting at 0)
<code>-(NSString*)substringFromIndex:(int)idx</code>	returns substring from the index to the end of the string

9.2 NSNumber

<code>+(id)numberWithBool:(BOOL)num</code>	
<code>+(id)numberWithChar:(char)num</code>	
<code>+(id)numberWithInt:(int)num</code>	
<code>+(id)numberWithLong:(long)num</code>	
<code>+(id)numberWithFloat:(float)num</code>	
<code>+(id)numberWithDouble:(double)num</code>	
<code>-(NSString*)stringValue</code>	return a textual representation of the number

9.3 NSArray, NSMutableArray

<code>+(id)arrayWithObjects:,...</code>	create a new array with the list of passed objects. The list must end with <code>,nil</code> .
<code>-(int)count</code>	returns count of objects in array
<code>-(id)objectAtIndex:(int)</code>	returns object at index (start counting at 0)
<code>-(NSEnumerator*)objectEnumerator</code>	return an enumerator (see 9.6) over all objects in array

NSMutableArray adds the following methods:

<code>-(void)addObject:(id)obj</code>	add the object to the end of the array
<code>-(void)insertObject:(id)obj atIndex:(int)idx</code>	insert new object before object with index <i>idx</i>
<code>-(void)removeObject:(id)obj</code>	remove all occurrences of object
<code>-(void)removeAllObjects</code>	remove all objects

9.4 NSMutableSet, NSMutableSet

Sets are bags which hold a finite number of objects.

<code>+(id)setWithArray:(NSArray*)arr</code>	creates a new set with all the objects in the array (see 9.3)
<code>+(id)setWithObjects:...</code>	creates a new set with all the list of passed objects. The list must end with <code>,nil</code> .
<code>-(BOOL)containsObject:(id)obj</code>	true if this set contains the given object
<code>-(int)count</code>	returns the number of objects in the set
<code>-(NSEnumerator*)objectEnumerator</code>	return an enumerator (see 9.6) over all objects in set
<code>-(NSArray*)allObjects</code>	return an array (see 9.3) of all objects in this set

NSMutableSet also implements the following methods:

<code>-(id)addObject:(id)obj</code>	add an object to the set
<code>-(void)removeObject:(id)obj</code>	remove object from the set
<code>-(void)removeAllObjects</code>	remove all objects from this set
<code>-(void)minusSet:(NSSet*)set2</code>	remove all objects from this set which are also in set2
<code>-(void)intersectSet:(NSSet*)set2</code>	keep only objects which are in both sets
<code>-(void)unionSet:(NSSet*)set2</code>	add all objects from set2 to this set

9.5 NSDictionary, NSMutableDictionary

Dictionaries are relations of values with keys. Objects are stored in the dictionary with a key. Later, we can access the object with the key. The search for matching objects with a key is quite efficient.

<code>+(id)dictionaryWithObjectsAndKeys:,...</code>	creates a new dictionary from the list of pairs (object, key). The list must end with <code>,nil,nil</code> .
<code>-(NSArray*)allKeys</code>	returns an array (see 9.3) of all keys in dictionary
<code>-(NSArray*)allValues</code>	returns an array (see 9.3) of all values in dictionary
<code>-(id)objectForKey:(id)key</code>	returns the value stored in the dictionary for key or nil if there is none
<code>-(NSEnumerator*)keyEnumerator</code>	return enumerator (see 9.6) over all keys
<code>-(NSEnumerator*)objectEnumerator</code>	returns enumerator (see 9.6) over all values

Class NSMutableDictionary also add the following methods:

<code>+(id)dictionaryWithCapacity:(int)cap</code>	creates a new dictionary with initial allocated space <i>cap</i>
<code>-(id)setObject:(id)obj forKey:(id)key</code>	store obj in dictionary for key
<code>-(void)removeObjectForKey:(id)key</code>	remove objects matching key
<code>-(void)removeAllObjects</code>	get rid of all pairs in dictionary

9.6 NSEnumerator

<code>-(NSArray*)allObjects</code>	returns an array (see 9.3) of all objects
<code>-(id)nextObject</code>	returns next object in enumeration or nil if we reached the end

9.7 Transcript

A representation for the user's console. You can write to this class' object or read from it.

This is actually provided by the StepTalk bundle of GNUstep.

<code>+(id)show:(id)obj</code>	prints <i>obj</i> or a string representation of <i>obj</i> to the Transcript
<code>+(id)showLine:(id)obj</code>	printf <i>obj</i> or a string representation of <i>obj</i> to the Transcript and advances to a new line
<code>+(NSString*)readLine: (NSString*)prompt</code>	prints <i>prompt</i> to the Transcript and accepts a line of input, which is returned

Index

- :=, 17
- add:, 12
- addAtom:, 8
- addHeterogen:, 7
- addObject:, 17, 25, 26
- addResidue:, 7, 11
- addScalar:, 13
- addSolvent:, 7
- alignmentBetween:and:, 10
- alignmentBetween:andChain:, 10
- alignmentPositions, 10
- alignTo:, 11
- allAtoms, 8
- allBondedAtoms, 9
- allChains, 6, 21
- allHeterogens, 7, 21
- allKeys, 18, 24, 26
- allObjects, 18, 21, 23, 26
- allResidues, 7, 21, 23
- allSolvent, 7, 21
- allValues, 26
- angleBetween:, 12, 24
- ARGS, 16, 20, 22
- arrayWithArray:, 21
- arrayWithObjects:, 25
- atDim:, 12
- atDim:value:, 12
- ATOM, 6, 9
- Atom, 9
- atomWithNumber:name:X:Y:Z:B:, 9
- atRow:col:, 12
- atRow:col:add:, 12
- atRow:col:divideBy:, 13
- atRow:col:multiplyBy:, 13
- atRow:col:subtract:, 12
- atRow:col:value:, 12
- attribute, 4

- block, 18
- bondTo:, 9
- boolean, 18

- C, 3
- calculateRMSD, 10
- centerOfMass, 13
- Chain, 6, 10
- chain, 5, 8
- chain1, 10
- chain2, 10
- ChainFactory, 6
- characterAtIndex:, 22
- charge, 9
- class, 3–5, 16, 17, 25
- close, 6, 15
- code, 6, 21

- cols, 12
- comment, 16
- COMPND, 6
- compound, 6, 21
- CompressedFileStream, 15
- console, 26
- containsObject:, 18, 26
- Coordinates, 9
- coordinates, 14
- coordsFromCoordinates:, 10
- coordsWithX:Y:Z:, 10
- count, 11, 18, 21, 22, 25, 26
- countHeterogens, 7
- countPairs, 10
- countPairsMaxDistance:, 10
- countResidues, 7
- countSolvent, 7
- countStandardAminoAcids, 7
- countUngappedPairs, 10

- date, 5, 21
- deriveStructuralAlignment, 10
- description, 6, 8, 10, 12
- dictionaryWithCapacity:, 26
- dictionaryWithObjectsAndKeys:, 26
- difference:, 11
- differenceTo:, 12, 24
- dimension, 12
- distanceCATo:, 8
- distanceTo:, 9, 21, 23
- distanceToLineFrom:to:, 9
- divideByScalar:, 13
- do:, 18
- doubleValue, 25
- dropAllBonds, 9
- dropBondTo:, 9

- eccode, 6, 21
- element, 9
- elementName, 9
- euklideanDistanceTo:, 12
- expdata, 6

- file
 - compressed, 15
- FileStream, 15
- findResiduesCloseTo:, 7
- floatValue, 25
- fromStreamAsTCoffee:, 10
- fullPDBCode, 6

- GCC, 3, 4
- get3DSequence, 7, 8
- getAtomWithInt:, 8
- getAtomWithName:, 8, 23
- getAtomWithNumber:, 8

- getCA, 8, 21, 23
- getChain:, 6, 23
- getHeterogen:, 7
- getOrigin, 14
- getResidue:, 7, 8
- getRotation, 14
- getSEQRES, 7
- getSequence, 7, 8
- getSolvent:, 7
- getTransformation, 10
- getTranslation, 14
- GNUstep, 4, 17, 25, 26
- gzip, 15

- haveAtomsPresent, 8
- HEADER, 5
- header, 5, 21
- HETATM, 6, 9
- HETNAME, 6
- hetnameForKey:, 6

- ifFalse:, 18
- ifTrue:, 18
- initFromString:, 13
- insertObject:atIndex:, 25
- interpreter, 3, 16
- intersectSet:, 26
- intValue, 21, 25
- invert, 13, 14
- isModified, 6, 8
- isNucleicAcid, 8
- isStandardAminoAcid, 8
- isTransposed, 12

- jacobianDiagonalizeWithMaxError:, 13

- key, 8
- keyEnumerator, 26
- KEYWDS, 6
- keywords, 6

- length, 12
- libmoltalk, 3, 4
- lifetime, 16
- linearizeTo:maxElements:, 12
- longValue, 25

- MacOSX, 25
- madd:, 13
- Matrix, 12
- Matrix44, 13
- Matrix53, 14
- matrixFromString:, 13, 14
- matrixIdentity, 13, 14
- matrixOfColumn:, 12
- matrixWithRows:cols:, 13
- message, 3, 16–18
- method, 3, 4, 16
- minusSet:, 26
- mixedProductBy:and:, 12, 24

- mmultiply:, 13
- moddescription, 8
- modname, 8
- MODRES, 6, 8
- MolTalk, 3–5, 16
- msubtract:, 13
- multiplyByScalar:, 13

- name, 6, 8, 9
- newChainWithCode:, 6
- newResidueWithNumber:name:, 7
- newResidueWithNumber:subcode:name, 7
- newStructure, 5
- newStructureFromPDBDirectory:, 5, 6, 21, 23
- newStructureFromPDBDirectory:options:, 5
- newStructureFromPDBFile:, 5
- newStructureFromPDBFile:options:, 5
- nextObject, 26
- NextStep, 25
- normalize, 12
- notNil, 18
- NSArray, 17, 25
- NSDictionary, 17, 18, 26
- NSEnumerator, 26
- NSMutableArray, 17, 21, 25
- NSMutableDictionary, 23, 26
- NSMutableSet, 26
- NSNumber, 19, 25
- NSSet, 17, 18, 26
- NSSString, 18, 25
- number, 8, 9
- numberCode, 6
- numberWithBool:, 25
- numberWithChar:, 25
- numberWithDouble:, 25
- numberWithFloat:, 25
- numberWithInt:, 25
- numberWithLong:, 25

- object, 3, 16–18
- object-oriented, 16
- objectAtIndex:, 18, 20–22, 25
- objectEnumerator, 25, 26
- objectForKey:, 18, 24, 26
- Objective-C, 3, 4, 17
- ok, 15
- oneLetterCode, 8
- optimize, 10
- origin, 10

- PairwiseStrxAlignment, 10
- parameter, 16–18
- PDB, 3
 - directory, 5
 - file, 6, 9
 - files, 3
- pdbcode, 5, 21
- PDBDIR, 5
- prepareResidueHash:, 7

- readLength:, 15
- readLine:, 26
- readLineLength:, 15
- readStringLineLength:, 15
- receiver, 17
- reference, 17
- REMARK, 6
- removeAllObjects, 25, 26
- removeChain:, 6
- removeObject:, 25, 26
- removeObjectForKey:, 26
- removeResidue:, 7, 11
- replaceObjectAtIndex:withObject:, 21
- Residue, 8
- ResidueFactory, 7, 8
- resolution, 6
- REVDAT, 5
- revdate, 5
- RMSD, 10
- rotateBy:, 9
- rotation, 14
- rows, 12

- scalarProductBy:, 12
- scaleByScalar:, 12
- scripting language, 3, 16
- selectedResidues, 11
- Selection, 10
- selectionWithChain:, 11
- SEQRES, 7
- sequenceNumber, 8
- setCharge:, 9
- setDimensions:, 12
- setElement:, 9
- setElementWithName:, 9
- setNumber:, 9
- setObject:forKey:, 23, 26
- setRows:cols:, 13
- setWithArray:, 26
- setWithObjects:, 26
- setX:Y:Z:, 9
- show:, 26
- showLine:, 18, 19, 21, 26
- Smalltalk, 3, 16, 17
- SOL, 6
- solvent, 6
- SOURCE, 6
- source, 6, 21
- square, 13
- StepTalk, 4, 16, 26
- Stream, 15
- streamAppendToFile:, 15
- streamFromFile:, 15
- streamToFile:, 6, 15
- stringByAppendingString:, 19, 25
- stringFromArray:, 18
- stringValue, 18, 21, 25
- stringWithCharArray:, 25
- stringWithFormat:, 25

- Structure, 5
- structure, 7
- StructureFactory, 5, 21, 23
- subcode, 8
- subtractScalar:, 13
- substringFromIndex:, 25
- substringToIndex:, 25
- sum, 13

- temperature, 9
- TITLE, 5
- title, 5, 21
- toStreamAsTCoffee:name1:name2:, 10
- toString, 12
- Transcript, 19, 26
- transformation, 10
- transformBy:, 7–9
- translation, 14
- transpose, 13

- union:, 11
- unionSet:, 26

- variable, 17
- Vector, 12
- vectorProductBy:, 12, 24
- vectorWithDimensions:, 12

- writeCString:, 15
- writeData:, 15
- writePDBToStream:, 6
- writeString:, 15

- x, 9
- x:, 13
- xIP:, 13

- y, 9

- z, 9