

Genomic Messaging System and DNA Mark-Up Language for Information-Based Personalized Medicine with Clinical and Proteome Research Applications

Barry Robson and Richard Mushlin

IBM Research, T.J. Watson Research Lab., Route 132, Yorktown Heights, New York 10598

Received December 31, 2003

The convergence of clinical medicine and the Life Sciences, commencing with opportunities in clinical trials and clinically linked medical research, presents many novel challenges. The Genomic Messaging System (GMS) described here was originally developed as a tool for assembling clinical genomic records of individual and collective patients, and was then generalized to become a flexible workflow component that will link clinical records to a variety of computational biology research tools, for research and ultimately for a more personalized, focused, and preventative healthcare system. Prominent among the applications linked are protein science applications, including the rapid automated modeling of patient proteins with their individual structural polymorphisms. In an initial study, GMS formed the basis of a fully automated system for modeling patient proteins with structural polymorphisms as a basis for drug selection and ultimately design on an individual patient basis.

Keywords: bioinformatics • genomics • sequence compression • DNA Mark-up • genomics language • personalized medicine • clinical messaging

1. Introduction

With the entry of medicine into the post-genomic era, bioinformatics is beginning to develop in the direction not only of the capability for enhanced pharmaceutical discovery,¹ but also for personalized medical treatment including selection and ultimately rapid design of therapeutics based on individual patient differences.² That humans differ by roughly 1–2 amino acids in an average protein sequence seems inevitable if we consider the amino acid as the minimum quantum of protein evolution, but it also implies the potential for a considerable variety in the individual patient responses to therapeutic drugs.² Unfortunately, it seems unlikely that a limited number of nucleotide loci can be held responsible for every possible genetic difference of pharmaceutical interest at the level of each individual patient. In consequence of this and for various research applications, early work³ foreseeing the need to store and transmit whole sequences or segments of sequences of human (and other) DNA sequences in an efficient manner, with due regard to correction of errors arising in information processing, is becoming timely. The early work focused on storage of sequences of amino acids in an efficient matter related to their chemical and evolutionary relationships, a “mode switching” structure to convert interpretation of strings of bits between amino acid and DNA and RNA sequences, and the reserved use of runs of identical bits (zero bits were chosen) in such a way as an error could be flagged if a stream of binary data was read out of phase.³ However, it went little further with management of nucleic acid sequences than to propose a corresponding two-bit code for each of A,G,C,T. Also, there were no capabilities to play a role in a clinical or biomedical

research infrastructure other than to accelerate searches and comparisons of protein sequences. 50 51

The present paper describes a much richer **Genomic Messaging System (GMS)** which deals with the transmission of clinical and biomedical data. It may be considered as a proposed specification for an approach with an emphasis on a specific language for embedding supporting information and management functions in streams of DNA data. Naturally, the details may evolve, but the concept of adding human and computer-generated content (such as annotation) into the DNA sequences, in a more general and powerful way, appears to be a useful one. The proof-of-concept code employs Perl 5 with capabilities for XML management although, as described below, the approach is not confined to XML-based records. Components have been also been recently encoded in Java. The description also includes the **Clinical Laboratory Messaging System (CLaMS)** as a straightforward modification in which the background genomic default usage is switched off so that it occurs at same rank level as other clinical data. GMS represents technology involved in the domain of information management of patient genomic information and associated clinical information. In particular, the current implementation is concerned with the compression, encryption, and transmission aspects of clinical and genomic data, including bringing data together such as the clinical record and the patient DNA results from the sequencing laboratory. The form of information transmitted is versatile and is capable of storing and transmitting an entire Integrated Medical Record (IMR). One may consider either that the transmitted stream is a “smart” DNA sequence or set of fragments containing rich annotation 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79

research articles

Robson and Mushlin

80 (potentially extracts from the patient records, and even medical
81 images), or simply that a whole medical record is streamed in
82 compressed form, in which DNA appears in the manner of
83 “clinical laboratory results”.

84 An unusual feature is that the GMS system is larger than,
85 and not confined to, an XML based approach. Whatever the
86 form of the input and final output, the original documents are
87 disassembled into a universal form known as “GMS Language”
88 and this language actually represents the form of the transmit-
89 ted stream. XML, for example, is disassembled and carried as
90 “annotation” within the data stream. More precisely, by “an-
91 notation” is meant any one from a set of possible data types
92 which can be embedded in a DNA sequence and certainly
93 include both annotation and XML types, but also many other
94 types including medical images and even new code which
95 executes as it is received. However, there are intrinsic tools to
96 assist in processing each data type and the XML tools are at
97 this time the most rich. Such processing includes reconstitution
98 of the XML on receipt. For example, an incoming XML
99 document in HL7 Clinical Document Architecture (CDA) could
100 be joined with incoming lab data and be reconstituted on
101 receipt as one unified XML document. The actual details of
102 application are defined by “plug-in” scripts called “cartridges”
103 that can readily be modified, so that the method is sufficiently
104 general that in principle microorganism, animal, and plant
105 information can also be managed. These can also include tools
106 to assist in legacy data conversion and in DNA and protein
107 annotation. The “plug-ins” then represent data conversion rule
108 files and the character and sophistication of data conversion
109 is a function of which of these files are loaded. In addition,
110 the cartridges can be changed such that the GMS system might
111 reappear several times in different functional roles at different
112 nodes in a life science workflow and information management
113 network.

114 Despite the above-mentioned potential broader application
115 outside of human medicine, a number of intrinsic tools such
116 as security features have been implemented that may be
117 considered peculiar to human medicine (though in fact they
118 would also be of value in, for example, protecting intellectual
119 property in regard to animal or plant breeding, or protein
120 engineering via microorganisms). These further needs arose
121 because clinical bioinformatics had certain different require-
122 ments to traditional bioinformatics. Traditional bioinformatics
123 addressed many species, and was more often no more fine-
124 grained than addressing a whole species. Clinical bioinformat-
125 ics, however, concentrates on the human species, on the
126 individual, and on that individual’s rights in relation, in
127 particular, to privacy and consent. Hence, for example, a
128 flexible system of passwords provides only selected parts of the
129 patient’s DNA sequences to be accessed by different research-
130 ers. There is also emphasis on the effect of genetic variation of
131 the details of gene products (proteins), their post-translational
132 modification and interactions. For example, when a message
133 specifying a patient’s DNA sequence is received by GMS, the
134 system adds automatic annotation to the DNA, translates the
135 DNA into protein sequences, and automatically annotates
136 those, while preserving the content and location of any existing
137 annotation sent with the original message.

138 A further feature less characteristic of traditional bioinfor-
139 matics systems is the ability to continue basic function, and
140 recover and operate on local devices should the supporting
141 information technology infrastructure fail or be destroyed in a
142 disaster. Since the future form of such a system may be

involved in a variety of medical care scenarios, including 143
emergency medical care, GMS has been designed to be 144
minimally dependent on other systems, when portability and 145
performance are paramount. The messaging network could 146
comprise direct communication between laptop computers or 147
other portable devices, without a server, and even the exchange 148
of floppy disk as the means of data transport. Basic tools for 149
reading unadorned text representation of the transmission is 150
built in, and could be used should all other interfaces fail. 151

2. Theory 152

2.1 Information Theoretic Basis. GMS does have a unified, 153
theoretical basis in information and communication theory, 154
but for the most part it serves to consider these within 155
descriptions of the method. In brief, there is due regard to 156
brevity and intrinsic error checking, using the notion only of 157
“current state”. Commands for the data management, data 158
items themselves, and optionally singlets, doublets, triplets, or 159
quadruplets of DNA sequence (a string of the characters AGCT) 160
each map to 8-bit bytes which are the fundamental units of 161
communications and storage. At present, files containing the 162
stream of bytes are sent and received. Writing and reading, and 163
actions which take place at the receiving screen and device, 164
take place byte by byte. This requires no stream memory save 165
current state, analogous to a Turing Machine; that is, a certain 166
limited number of GMS state variables are set when encounter- 167
ing commands in the stream, and the stream is never “re- 168
wound” to reexamine earlier information. Hence, this is equally 169
applicable to storage, or direct transmission and reception, with 170
action byte by byte. Further, to meet demands of very high 171
security it is a simple matter to modify the code so that bytes 172
are destroyed immediately after being read, without losing the 173
precise effect of the stream content. Though this concept was 174
originally intended to illustrate what we mean by “current 175
state”, it has now been appreciated that, apart from issues of 176
security and privacy, a patient could give consent conditional 177
on his data being used just once or a specified number of times. 178
The fact that the destruction of the data is so intrinsic to its 179
utilization may be reassuring to the patient. 180

2.2 Architectural Theory. We here take the position that the 181
overall approach reflected in the architecture is a theoretical 182
aspect, the plan reflecting our philosophical position in GMS 183
development, and that its actual realization is in terms of 184
content representation, most notably the GMS language (GMSL). 185
The most important theoretical concept related to the 186
architecture is that GMS is a “split node” (it is GMSL which 187
realizes the mediation between the two halves of the node). 188
Above it was mentioned that GMS can be used at multiple 189
nodes of a life sciences workflow and information management 190
network, and that special attention has been given to continued 191
operation and recovery should the supporting infrastructure 192
fail. To do this, GMS is split into a data receive or capture part, 193
a storage and/or transmission part, and a receive, automatically 194
annotate, and process part. Should the infrastructure fail, the 195
information being processed by each node will normally be 196
safely stored in an encrypted form. 197

The method may be conveyed top-down, starting from a 198
systems architecture perspective. The basic one-node config- 199
uration (i.e., comprising two halves of one “split node” con- 200
figuration) is described. The communications (and storage) 201
component will be considered first because features related to 202
efficiency, security, privacy, accuracy, and authentication are 203
built into the core components and thus serve as a basis for 204

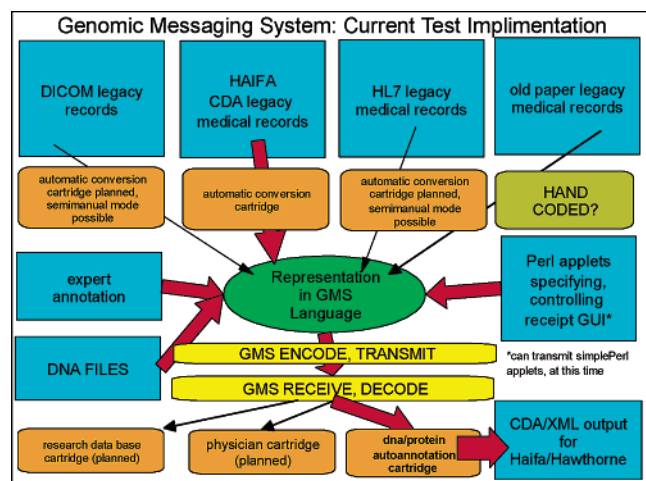


Figure 1. Encoding process, including the option of encrypting the binary stream using a scrambling algorithm. A key is required at the receiving end to unscramble the stream. Future implementations may allow different parts of the data to be encrypted using different keys.

dealing with these issues in the larger system. Figure 1 gives an overview of how GMS fits into the world of heterogeneous, autonomous, loosely coupled systems of clinical and genomic data. The large blue boxes at the top show examples of clinical data repositories, conforming to varying degrees of standardization. The green oval in the center shows the data extracted from the repositories converted to GMS representation. The arrows leading from the repositories to the GMS representation are labeled with possible mechanisms for handling the conversion. The current prototype is focused on converting records from the IBM Haifa repository, which conform to the HL7-proposed Clinical Document Architecture (CDA) standard (hence, the large red arrow).

The concept of "plug-ins" or "cartridges" is important theoretically to distinguish the universal core features of the "split node", from the addable features which give different functions at different node points or simply of one GMS unit in different application contexts. In Figure 1, the conversion to GMS syntax is done automatically by one of several such possible GMS "cartridges". These cartridges are adapters used by GMS to perform various input/output conversion tasks. In the current prototype the cartridges are written in Perl and are activated by the main GMS Perl program. In the future GMS may support other cartridge implementations, such as XSLT.

The reference to "Haifa" and "Hawthorne" represents the recent configuration spread across various IBM Research sites of which the above two were principal collaborators in Israel and Westchester New York respectively, GMS being primarily developed and executed at the Yorktown Heights site. Each of these three sites was however variously in communication with collaborating hospitals. In particular, the scheme was intended to demonstrate transmission of data between real hospitals at internationally separate sites.

Data from genomic databases is brought into GMS via files which contain the DNA raw sequences and optionally, but importantly, allow annotation by an expert. This process is indicated schematically at the left of the figure by the two blue boxes and two red arrows feeding the GMS representation. In the current implementation, the expert annotates the DNA files directly with a text editor, and the modified DNA files are then automatically converted into GMS syntax. The syntax of the

DNA files prior to conversion is quite flexible and supports XML tags for annotation plus special GMS commands for process control.

An additional input is provided for packaging an application with the data. On receipt of the GMS package, the program is optionally extracted and run. The current implementation supports sending simple Perl programs for displaying the data. This feature is shown at the right of the figure.

Once all the inputs are represented in GMS syntax, the encoding process begins. This process is represented by the upper yellow bar in the figure. The GMS encoding process is a stream-oriented algorithm which results in a compact binary representation of the combined input data. (A detailed description of the encoding process is given in a later section.) The compact binary stream is optimized for compression of DNA sequences. Unlike the "natural" languages used for clinical data and program source code, DNA contains seemingly random distributions of characters, and so, being "information rich" by virtue of its distribution of characters, it does not compress well with standard techniques. GMS uses as few as 2 bits per base, with the theoretical limit for a 4-character alphabet.

At present, the receiver must satisfy fairly stringent requirements if maximum options for security are set. In the most demanding scenario, the receiver must know the number representing the level of encryption chosen, two tumbler key values associated with that encryption, a patient identifier in explicit or encrypted form, the nature of the terminator signal used to terminate partitions of the data, and, in some modes of use, whether text compression was applied to either upper or lower case, and every password and file-lock specifically coded in the system. Further, she or he may be obliged to have at hand a "template" or "filler" file or files which complete the patient's DNA data. The above features may be considered extra due diligence in regard to privacy because, in addition to the above, state-of-the-art industry standard encryption can be used to encode and decode the stream, between the encoding and decoding steps performed by the GMS system.

The encoded, compressed, encrypted binary stream is output into a file on the transmitting system. In this state the stream can be transmitted by any means to the receiving system, where the decoding process takes place. This process is represented by the lower yellow bar in the figure. The decoding process decrypts and uncompresses the stream into the GMS internal representation, and activates one or more output cartridges to produce data in a form compatible with various applications.

The output cartridges are shown in orange at the bottom of the figure. The current prototype includes one such cartridge for translation of portions of the DNA sequence into amino acid sequences, followed by the automatic annotation of the resulting proteins. The annotated GMS representation is then converted into an XML file for use by other applications. It is intended that this XML output file be as CDA¹ compliant as possible. Other output cartridges envisioned at this time include one tailored for physician use and another for research use, in which the identity of the individual is kept hidden. In addition to the cartridge outputs, the system also produces a "dump" of the GMS representation for use in debugging.

It is useful to group the components of GMS into two categories: "core" components and "cartridge" components. The core performs the encode-, compress-, encrypt-, decrypt-, uncompress-, decode-cycle which begins and ends with all the data in the GMS representation. The implementation of this core, including the GMS syntax, is now stable, with enhance-

246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308

research articles

Robson and Mushlin

309 ments limited to performance enhancements and some new
310 features. The core components are implemented as a single
311 Perl executable program, with the choice between send and
312 receive function determined by a runtime parameter. Once that
313 choice is made, additional parameters control input selection
314 and encryption for the send function, and decryption and
315 output selection for the receive function. It is envisioned that
316 the send function would be used by the “owner” of the data to
317 be sent, as well as trusted parties such as the clinical bioin-
318 formaticist or referring physician. The receive function might
319 be used by a consulting physician or researcher.

320 The cartridge components operate on the data either before
321 or after processing by the core components. Examples of
322 cartridges for converting input and output have been men-
323 tioned above. The current prototype implements cartridges as
324 Perl scripts containing rules expressed using regular expres-
325 sions. These scripts are executed at runtime under the control
326 of the core program but are packaged as separate files. Some
327 of these cartridges, such as the protein annotation example,
328 might be considered mini-expert systems. It is envisioned that
329 these cartridges will evolve to become considerably more
330 “expert”, and proliferate to accommodate a wider variety of
331 inputs and outputs as GMS development and deployment
332 proceeds. A mechanism for supporting more loosely coupled
333 cartridges, written in other languages, is being discussed. As
334 the cartridge functionality grows, the distinction between a
335 cartridge and an application begins to blur. The cartridge
336 concept is currently limited to processing functions requiring
337 little or no user interaction.

3. Method

339 **3.1 Genomic Messaging System Language (GMSL).** The core
340 components consist of the GMS language (GMSL) and the Perl
341 program which processes data represented in this language
342 according to instructions included with the data and runtime
343 parameters. GMSL is a novel “lingua franca” for representing
344 a potentially broad assortment of clinical and genomic data
345 for secure and compact transmission using the Genomic
346 Messaging System. The data may come from a variety of
347 sources, in different formats, and be destined for use in a wide
348 range of downstream applications. However, GMSL is opti-
349 mized for annotation of genomic data. This section explains
350 the basic features of GMSL. A full description of the syntax and
351 usage is provided in the Appendix.

352 **3.1.1 Primary Functions.** The primary functions of this
353 language are to accomplish the following:

354 ✓ Retain such content of the source clinical documents as
355 are required, and to combine patient DNA sequences or
356 fragments.

357 ✓ Allow the expert to add annotation to the DNA and clinical
358 data prior to its storage or transmission.

359 ✓ Enable addition of passwords and file protections.

360 ✓ Provide tools for levels of reversible and irreversible
361 “scrubbing” (anonymization) of the patient ID etc.

362 ✓ Prevent addition of erroneous DNA and other lab data to
363 wrong patient record.

364 ✓ Enable various forms of compression and encryption at
365 various levels, which can be supplemented by standard meth-
366 ods applied to the final file(s).

367 ✓ Select methods of portrayal of the final information by
368 the receiver, including choice of what can be seen.

369 ✓ Allow a special form of XML-compliant “staggered”
370 bracketing to encode DNA and protein features which, unlike
371 valid XML tags, can overlap.

These functions are all implemented in the current proto- 372
type. 373

3.1.2. Byte Mapping Principle. GMSL, like most computer 374
languages, recognizes two basic kinds of elements: instructions 375
(commands) and data. Since GMS is optimized for handling 376
potentially very large DNA sequences, the structure of these 377
elements is designed to be compact. The unit of processing is 378
an 8-bit byte, but the individual bits have significance as 379
commands or data. This approach allows some bytes to be pure 380
commands, some to be pure data, and some to be part 381
command and part data. (Such so-called “mixed commands” 382
are actually commands which implicitly “represent” or invoke 383
data). 384

The mixed type is motivated by the fact that any of the four 385
DNA bases can be represented using only 2 bits. Therefore, 386
sequences of 1 (singlet or SNP), 2 (doublet), or 3 (triplet) bases 387
can fit in a single byte, with 2 or more bits left to distinguish 388
between the possible arrangements. Shown below are the four 389
mixed commands for representing these cases. 390

Case	Bits
Singlet (A, C, G, or T)	XX000100
SNP (any 1 base)	XX100100
Doublet (any 2 bases)	XXYY1100
Triplet (any 3 bases)	XXYYZZ11

In this table, the 0's and 1's represent the commands, and 391
the Xs, Ys, and Zs represent the 1, 2, or 3 bases expressed as 2 392
bits each, as follows 393

$$A = 00 \quad G = 01 \quad C = 10 \quad T = 11$$

It is convenient that this scheme allows a codon for an amino 394
acid to be represented by the triplet case. It may also be useful 395
in terms of the performance of downstream applications, that 396
the complementary bases are the binary negations of each 397
other, that the left bit distinguishes purine (0) from pyrimidine 398
(1), and that bases with both bits identical (00, 11) have the 399
stronger, triple hydrogen bonds, whereas those with different 400
bits (01, 10) have weaker double hydrogen bonds. 401

Another class of command that relates to the byte mapping 402
principle allows 4 bases to be packed into a single byte to give 403
the most compressed stream. This feature is useful for handling 404
long DNA sequences uninterrupted by annotation. The tight 405
packing continues for a preset distance, or up to a statistically 406
unlikely terminator. This compressed data can either be 407
transmitted in the main stream, or read from separate files 408
during the decode process. These genomic “background” files 409
are a key ingredient in the GMS security scheme. By transmit- 410
ting the genetic “base” and “variations” separately, the indi- 411
vidual's specific DNA sequence is optionally obscured from all 412
except those having access to both parts. 413

Another type of mixed command is used to open or close a 414
“bracket”, like parentheses, for grouping data together. Unlike 415
parentheses, or markup tags, which can only be “nested”: 416
{a[b(c)d]e}, GMS brackets can be crossed: {a[b(c)d]e}. This 417
feature is required for genomic annotation because regions of 418
interest often overlap. The mixed commands for brackets are 419
as follows: 420

Open bracket xxxxxx01

Close bracket xxxxxx10

where the 6-bit quantity “xxxxxx” is one of 64 possible bracket 421
types, analogous to “round”, “square”, “curly”, etc. 422

Genomic Messaging System

423 In addition to these “mixed” commands, there are com- 471
 424 mands which are not associated with any data, as well as 472
 425 commands which are associated with a number of bytes of 473
 426 data. Other than having the form XXXXX000, these commands 474
 427 have relatively arbitrary bit patterns, and so are not discussed 475
 428 in this section. The use of these commands will be described 476
 429 in detail in the sections that follow. In total, there are 256 477
 430 possible command bytes, most of which have been assigned a 478
 431 function in the current prototype. One command is reserved 479
 432 for switching to a 2-byte command set for future expansion. A 480
 433 special meaning is assigned to the “null byte” 00000000, 481
 434 corresponding to the command “warning” or danger. Because 482
 435 of the way the command bit patterns have been assigned in 483
 436 the encode process, this particular bit pattern should never be 484
 437 encountered in uncompressed data outside of a data statement, 485
 438 and if it is, GMS will terminate with warnings. This feature is 486
 439 described below.

440 **3.2 GMS Command Usage.** The primary data is seen as DNA 487
 441 base sequences (strings of symbols GCTA) to be compressed 488
 442 three-to-a-byte. This triplet encoding maps directly to the 489
 443 amino acids of protein sequences. Specific encoding into one 490
 444 or two symbols per byte can specifically be done by separating 491
 445 the symbols by semicolons or newlines. In the example 492

AGGC;TT;AGCCT

446 the TT will be stored as a doublet in one byte, the rest as triplets 495
 447 as much as possible. With this field of DNA data, commands 496
 448 and generic DATA statements (see below) can appear separated 497
 449 by semicolons or newlines, and commands of multiple words 498
 450 are tolerant to placement and whitespace. 499

451 **3.2.1 Example Commands.** Some command examples are 500
 452 listed here to give a flavor for the GMS language (terminal 501
 453 semicolons are not necessary at end of line). The full command 502
 454 set is shown in Appendix 1.

read in dna;

455 which reads in DNA from the specified file and compresses it 503
 456 2 bits per base character 504

validate;

457 which may appear to represent and count validation points to 505
 458 ensure that they have been seen exactly *N* times, or 506

danger;

459 which is normally wrapped in another environment and should 507
 460 never be seen deliberately. Two danger commands in succes- 508
 461 sion will guarantee detecting a phase shift of incoming data 509
 462 by 1–7 bits. (The “danger” command is encoded by 00000000 510
 463 which is also the danger signal, and this string of bits will always 511
 464 be encountered by a phase shifted read encountering 512
 465 0000000000000000).

466 A sample command phrase or group made up of several 513
 467 commands:

```
password;[&7aDfx/b{by shaman protect data};
xml;<gms:{patient}_dna>];index;and protein;
filename[template.gms{by shaman unlock data}];read in dna
xml;</gms:{patient}_dna>];index;and protein;
```

468 Here, the command **password** in the command phrase 514
 469 **password;[&7aDfx/b{by shaman protect data}**, which will 515
 470 allow the incoming stream to be read and active from that point 516

only if (a) the receiver has already entered a patient ID which 471
 encrypts to &7aDfx/b, and (b) if at that point the 472
 receiver enters another password, here **shaman**. Data item 473
filename;[template.gms{by shaman unlock data}] allows the 474
 data of the file specified to be incorporated into the stream 475
 only if that password, here **shaman**, was the last entered, 476
 helping ensure that the correct file is loaded and ensuring that 477
 the field has not been intercepted and falsely continued by a 478
 hostile agent. Another and further password command, with a 479
 different password requested, could follow the first password 480
 request. The dna on file in any event is effectively a filler 481
 template between the polymorphisms, and might be transmit- 482
 ted only once to the physician, so serving as a further kind of 483
 encryption key in the general sense. The **xml** command 484
 requests annotation in xml format: the tag names are calcul- 485
 ated in part on receipt of the stream from the variable 486
{patient}, specified elsewhere and here carrying the patient’s 487
 name. This will be “scrubbed” into an encrypted form if an 488
 earlier specification was set. The xml tags are shipped with the 489
 stream. The command **index** includes them as tags in the 490
 output CDA XML file, and command **and protein** requests that 491
 this annotation is interlaced with the automated annotation 492
 of the DNA and of the resulting protein sequences explored in 493
 all six reading frames. 494

A valuable DNA/protein annotation command is of the 495
 example form 496

(43

which forces onto the final xml output file the tag e.g., <open 497
 feature=“whatever” type =“43” level=8/> depending on the 498
 bracket level, and is used to annotate overlapping features DNA 499
 and protein features which are impermissible to simple use of 500
 XML (in the sense that to XML <A> is permissible, <A> 501
 is not). 502

Generic DATA statements encode specific or general classes 503
 of data which include: 504

```
data ;[...../];
password ;[...../];
filename;[...../];
number ;[...../];
xml;[...../]; (XML)
perl;[.....{end of data}] (Perl applet executed on receipt)
hl7;[.....{end of data}] (HL7 messages)
dicom;[.....{end of data}] (images)
protein ;[...../];
squeeze dna;*....../ (compress DNA to 4 characters per byte.)
```

Alternative forms such as data;/...../ are possible. The 505
 terminating bracket ‘]’ is optional and is actually a command 506
 to parity check the contents of the data statement on receipt. 507
 Within the fields [..... can be inserted text 508
 permitted by the “type”. Type restriction is currently weak, but 509
 backslash would be prohibited in certain types of data to avoid 510
 the fact that it is a permissible symbol in content. 511

A wide variety of commands in curly brackets (French 512
 braces) can appear in these DATA fields, such as {xml symbols}, 513
 {define data}, {recall data}, {on password unlock data}, or 514
 carry-variable names such as {locus} which are evaluated and 515
 macro-substituted into the data only on receipt. 516

3.2.2 Example Common Command Phrases and Recipes. 517
 Appendix 1 includes a number of these. The basic language 518
 can be used to make countless phrases out of their combina- 519

research articles

Robson and Mushlin

520 tions, but there are relatively few complex commands formed,
521 such as

```
filedata;[{by shaman unlock data}]
number;[15 base pairs\]
squeeze dna
*
AGCTTCAGAGCTGCT\
```

522 which places a protective lock on the following data, requiring
523 password “shaman” for access, compresses 15 base pairs of
524 DNA into four base pairs per byte as much as possible. Another
525 example is:

```
name;[mary\];xml;[elizabeth {define data}]
xml;[<test> patient {identifier} has informal code name {mary}</test>\];index
```

526 which illustrates both use of the use-defined variable “mary”
527 and the system variable “identifier” (the current patient identi-
528 fier) in writing specifically stated XML (the *<test>* tags and their
529 content).

530 **3.2.3 Advanced Considerations for Legacy Input.** To facili-
531 tate utilization of existing DNA files with little or no modifica-
532 tion, the following applies to the pilot version. At present, lower
533 case **a,c,g,t** or other symbols such as **U** or **u** must be converted
534 to A, G, C, T prior to input. Automation of this conversion,
535 including a fuller treatment of ambiguity symbols such as **M**
536 for A OR C, is currently under research and development in
537 the Java version. The ambiguity codes for DNA bases are a
538 longer term project since their manipulation and representation
539 as interpreted and annotated protein sequences, is a matter of
540 some debate, and, most generally, the use of ambiguous entries
541 are discouraged for clinical usage: rather, the DNA is simply
542 not entered, and the rest of the DNA in that region is entered
543 as a fragment, in a manner indicating that it is incomplete.
544 Such strings of symbols A, G, C, T with or without the
545 semicolons, are valid GMSL without qualification, save that
546 addition of the “end of task” command is preferred for
547 validation of correct termination (see below). Among valid DNA
548 as a string of characters A, G, C, T, more lavish and powerful
549 GMSL commands, described below, can be interspersed. Then,
550 anything not corresponding to A, G, C, or T, or to a recognized
551 command, is taken as comment and is ignored. As far as the
552 recognized commands are concerned, the philosophy used is
553 that GMS commands can be considered as DNA message
554 content and an inbuilt protocol for managing the DNA during
555 transmission and storage of the DNA stream, and hence that
556 these commands “embellish” the widely accepted use of A, G,
557 C, T streams as input.

558 The above should not concern the routine user: the correct
559 usage and action will be represented in well tested scripts with
560 GMS validation checks, in contexts outlined here. Similarly, the
561 following consideration in this section is in regard to particu-
562 larly advanced usage, included here for completeness of general
563 description of command usage.

564 Where else can GMS commands appear? Without conversion
565 cartridges to convert legacy data, the default input is
566 DNA+GMSL as above, i.e., comprising both A, G, C, T symbols
567 and the commands described below. The question of other
568 input, as long as one is discussing the default mode, does not
569 then normally arise. However, the primary function of GMSL
570 is as a Lingua Franca into which legacy documents are
571 interpreted. In routine use such as in handling legacy docu-
572 ments, typically as clinical records and laboratory returns, the
573 GMS commands are of course not written by the routine user

but invisibly by conversion cartridges with the .rex suffix (see 574
above). In that more prevalent mode, in contrast to the default 575
use above, GMS commands do not routinely appear in the 576
input in the form of such incoming documents. So, typically, 577
it is recommended that GMSL does not appear in incoming 578
documents and that its generation is handled by the scripts 579
within the cartridges. 580

That being said, there is in fact nothing to stop GMS 581
commands from being mixed in with DNA data when the DNA 582
enters the stream in legacy or laboratory documents, in the 583
above philosophy that DNA data can be “embellished” DNA 584
data. For example, within a document from the clinical DNA 585
lab, there might be scattered among the DNA some GMS 586
commands which highlight polymorphisms or deletions, or use 587
of a command which instructs that part of the DNA be 588
squeezed with maximum compression of 2 bits per base pair 589
(i.e., 4 base pairs per byte of 8 bits). The highlighted text below 590
illustrates GMS embedded as content in a fragment of an XML 591
document from a clinical DNA laboratory. 592

```
</gms:t_cell_epitopes>
</gms:nonsequence_annotation>
```

```
filedata;[{by shaman unlock data}]
number;[87 base pairs\]
squeeze dna
*
```

```
ATGGTGTGTC TGAAGCTCCC TGGAGGCTCC TGCATGACAG CGCTGACAGT
GACACTGATG GTGCTGAGCT CCCCACTGGC TTTGGCT\
```

```
<gms:experimental_start_of_mature_peptide>

GGG GAC ACCCGA
<gms:template_start>

<gms:snp>
G
<gms:annotation>
allotype is g in drb1*0101, r in drb1*0105
</gms:annotation>
</gms:snp>
```

593 However, in the pilot version, some simple considerations are
594 currently required to satisfy international XML compliance
595 within the context of GMS error-checking and the “embellished
596 DNA stream” philosophy. This applies in the case of existing
597 conversion cartridges processing files using .gmi and .gmd
598 extensions files (see below), which are concerned with XML
599 adhering to HL7 Inc.’s Clinical Document Architecture. It would
600 also apply in the case of a syntactically invalid XML document
601 which might be processed to syntactically correct XML by
602 future cartridges or which becomes a valid XML document
603 when spliced in with other input. Curly braces {..} in GMSL
604 commands within XML contexts should be replaced by *(and)*
605 respectively. This is because {..} are not approved basic XML
606 content; in fact, to be compliant with XML recommendations,
607 the existing conversion cartridges automatically convert all
608 occurrences of ‘{’ and ‘}’ to their XML-approved &# formats
609 { and } respectively, unless they are protected by the
610 *(...)* alternatives so that the GMS parser can ultimately read
611 them. Subsequent conversion of the *(...)* to {...} forms so that
612 the GMS parser can read them is also a function of the.rex
613 legacy conversion cartridge for managing XML documents.

614 **3.3 GMS Encoder/Decoder Module.** The sending (or encod-
615 ing) and receiving (or decoding) components are represented
616 in the same program. The choice of decoding means that the
617 user can see only the stream of bytes which has been received
618 on a “gms stream” file. Encoding will invoke both encoding
619 and a test decoding under full receiving conditions, guarantee-
620 619

620 ing that, if the receiver has the same software and file, and
621 choice of passwords etc., he will be able to interpret the file
622 correctly and will see the effects at the screen intended by the
623 encoder (see next section). Since the information stream sent
624 can include Perl code and Perl applets, the identical code
625 version can be sent. At decode time, not only routine (e.g.,
626 parity) checks are performed, but all data in memory and in
627 files created by the encoding and decoding steps are compared
628 byte for byte.

629 The GMS language file containing commands and data as
630 described previously is compressed into a messaging stream
631 which may be optionally further compressed and encrypted.
632 Various options are available when starting the program.

633 **3.3.1 Mode and Filename Option.** Choice of encode (e) or
634 decode (d) mode, plus choice of "root" name for the files used
635 if the default name 'stream' is not satisfactory. There is also at
636 this point an optional template (t) mode: in this mode the
637 system immediately walks the user through a master template
638 example file asking the user to enter replacement data for the
639 character string "???" whenever encountered.

640 **3.3.2 Privacy Option.** Choice of whether the received docu-
641 ment will contain the actual patient identification, or be
642 "scrubbed" by replacing it with an encrypted identifier.

643 **3.3.3 Case Selection Option.** Choice of compression of either
644 upper or lower case text into two characters per byte. Although
645 this can save memory for extensive text, another function is to
646 render unauthorized decryption more difficult.

647 **3.2.4 Terminator Selection Option.** Option to replace part
648 of the sequence of characters which represent a 'terminator',
649 i.e., signifies the end of transmission of particular data field
650 within the transmitted stream. The default is a limited number
651 of terminators, some with command-like functions, of which
652 {end of data} is the most basic, and all of which end in ...data}.
653 This option allows the word 'data' to be replaced with a string
654 of any length to ensure a terminator which is unlikely to be
655 encountered by chance even in very large amounts of data,
656 e.g., transforming the terminator {end of data} to {end of a
657 very large amount of stuff}. The a priori probability of this
658 chance happening is reported when this option is used. No
659 input files are permanently changed by this choice, but the
660 receiver must know the choice.

661 **3.3.5 Encryption Level Option.** This determines choice of
662 level of encryption if any. The byte stream will be shuffled the
663 specified number of times following a machine-dependent
664 "random number" generator linked to the iteration count of
665 the generation to avoid recycling. Must be same to decode.

666 **3.3.6 Tumbler One Option.** "Tumbler One": Choice of
667 number seed which may be, e.g., the Social Security Number.
668 Must be same to decode.

669 **3.3.7 Tumbler Two Option.** "Tumbler Two": Choice of size
670 of blocks of bytes which may be shuffled (larger sizes speed
671 encryption process). Must be same to decode.

672 **3.3.8 Identifier Option.** Identifier: The text input here not
673 only affects the encryption but on receipt is compared with
674 an encrypted string in the byte stream being decoded. Must
675 be the same to decode, on both counts.

676 **3.4 Input.** As described in the overview, GMS incorporates
677 data from three different inputs before processing the data into
678 a binary stream. The first input is the "clinical" data, which
679 can come from a variety of sources, but is expected to be CDA
680 compliant in the preferred implementation. The second input
681 is "genomic" data, also from any source, in the form of DNA
682 bases, perhaps with annotation. This input must be prepro-

683 cessed by inserting a minimal number of GMS commands. The
684 third, and optional, input is a Perl program to be transmitted
685 with the data and run on receipt. Interfacing GMS to clinical
686 and genomic data management systems involves adapting
687 existing data sources to the input requirements of GMS.

688 Note that these input routes can be bypassed by directly
689 entering data into the primary GMS input file using GMSL
690 syntax. This "free-form" approach involves the annotation of
691 a DNA sequence or other clinical data, and instructions on how
692 it is to be transmitted and seen by the receiver. This manual
693 process is not considered as using a legacy conversion cartridge,
694 though there are, under early development, various support
695 tools to facilitate data preparation. At present, this mode is used
696 primarily for testing.

697 **3.4.1 Input File Conventions.** As mentioned in the section
698 on Options, the user selects a root filename (no extension) to
699 be used for a complete GMS run. The extensions, or suffixes,
700 for the various files are set by GMS convention. The primary
701 input file is a **.gmi** (genomic messaging input) file, from which
702 GMS automatically generates a **.gms** (genomic messaging
703 system) file containing GMS commands and data expressed
704 in the canonical GMSL representation. It is useful to think of
705 the GMSL canonical representation (the **.gms** file) as (a) the
706 final form which GMS uses to generate the encoded byte stream
707 as a **.gmb** (genomic messaging binary) file, and (b) as the
708 "lingua franca" and "Grand Central Station" for bringing
709 together incoming data, from the primary input file (the **.gmi**
710 file) and any secondary input files (described below).

711 In cases of direct input, the user manually creates the content
712 of the **.gmi** file. The appearance of this file will be very similar
713 to the **.gms** file which is generated from it and which is
714 converted to the stream of transmitted bytes. There are three
715 exceptions at present because they are affected by options
716 requested on startup of the GMS system when the prepared
717 **.gmi** file is first processed:

718 i. Text compression of upper or lower case text takes place
719 in the generation of the **.gms** from the **.gmi** file.

720 ii. The string {identifier} on the **.gmi** file can be used to stand
721 for the identifier of a specific or generic patient which option-
722 ally, is only made explicit, when the **.gms** file is constructed or
723 (when the "scrub" option is chosen) on receipt of the GMS
724 message.

725 iii. The occurrences of the terminator word data in the
726 terminator signals {...data} can be altered in generating the
727 **.gms** file.

728 If none of these facilities are used, however, the **.gms** and
729 **.gmi** files would be identical for the direct input case.

730 A built-in variant of this approach is always accessible as a
731 support tool. The user can automatically generate the internal
732 image of a **.gmi** file by invoking the template (t) mode of startup
733 of GMS, in which case the system will walk the user through a
734 template stored in a **.gmt** (genomic messaging template) file,
735 requesting where new variable data is to be added (which is
736 whenever characters "???" are encountered in the template file).
737 The result is subject to the above transformations which
738 normally occur when a **.gmi** file is converted to a **.gms** file.

739 In many cases, it is more convenient to think of the **.gmi**
740 file as the clinical context to which genomic data will be added.
741 In such cases, the genomic data, represented as DNA sequences
742 with optional GMS commands and data included, is taken from
743 a **.gmd** (genomic messaging dna) file.

744 **3.4.2 Clinical Data Input.** The clinical data input file
745 provides the skeleton structure into which the annotated

research articles

Robson and Mushlin

746 genomic data will be merged. The preferred implementation
747 requires that the clinical input file be CDA compliant, although
748 GMS could accept any text file in this role, and the current
749 prototype does not enforce compliance. The CDA file structure
750 can be complex, but for the purpose of GMS, only the basic
751 features are important, and will be described here.

752 The CDA file is a well-formed XML document. The root
753 element corresponds to the "level" of CDA compliance. CDA
754 <levelone> is the least restrictive, <levelthree> is the most
755 restrictive. The current prototype assumes only <levelone>
756 structures. Within the CDA document is exactly one
757 <clinical_document_header> and exactly one <body>. The
758 header structure is specified in very deep detail by CDA, but
759 for current GMS purposes, it serves primarily to identify the
760 patient, and is processed essentially verbatim. The body
761 structure is more flexible than the header. It contains the
762 clinical content expressed using a small number of CDA-
763 defined structures. GMS merges the genomic data into the body
764 using these same structures. Shown below is an example of a
765 clinical data input file, showing only the outermost XML
766 structures for use with the current prototype. An example of a
767 clinical CDA file obtained from SHAMAN collaborators in Haifa
768 for a bone marrow transplant case is shown in its entirety in
769 Appendix 2.

```
<levelone>
  <clinical_document_header>
    <!--header structures per CDA-->
  </clinical_document_header>
  <body>
    <!--clinical content per CDA-->
    <!--GMS merges genomic data here-->
  </body>
</levelone>
```

770 Currently, GMS ignores the meaning of the clinical content.
771 In the future, expert system cartridges or downstream applica-
772 tions will add clinical and genomic annotation based on the
773 availability of the combined clinical-genomic context. This is
774 one of the principal paradigm shifts that the SHAMAN project
775 is trying in general to encourage, and that GMS in particular
776 directly enables.

777 **3.4.3 Genomic Data Input.** The genomic data input file
778 (.gmd) contains the DNA sequences and optional manual
779 annotation. The DNA sequences are strings of bases. Whitespace
780 is ignored. The annotation is inserted using XML-style tags with
781 a "gms" prefix, but the file is not an XML document. Currently,
782 the minimum annotation required for GMS to correctly process
783 the file is <gms:genomic_data> as the root element. An
784 example of a genomic input file is shown in Appendix 3.

785 **3.4.4 CDA Legacy Conversion Cartridge.** The "cartridges"
786 are replaceable program modules which transform input and
787 output in various specialist ways. They may be considered as
788 mini "Expert Systems" in the sense that they script expertise,
789 customizations and preferences. All input cartridges ultimately
790 generate .gms files as the final and main input step. This file is
791 converted to a binary .gmb file and stored or transmitted. Input
792 cartridges include at this time primarily the Legacy Conversion
793 Cartridges, for conversion of legacy clinical and genomic data
794 into GMS language, of which in turn the CDA (Clinical Docu-
795 ment Architecture) cartridge is the best established at this time.

796 When the .gmi file is a CDA document (as might be expected
797 when retrieving data from a modern clinical repository), GMS
798 needs to know how to convert the content, marked up with
799 CDA tags, into the required canonical .gms form. This is
800 accomplished using a GMS "cartridge". In this scenario rep-

resenting the first GMS cartridge application supporting au- 801
tomation, the expert (optionally) modifies a file obtained in 802
CDA format to include additional annotation and structure. 803
Again, the template mode described above is available to help 804
guide this process so that the whole modified document 805
remains CDA compliant. The resulting CDA document with 806
added genomic features represents a "CDA Genomics Docu- 807
ment". Such a CDA document can now be automatically 808
converted into GMSL. 809

This conversion cartridge for CDA formats is the first type 810
of "legacy record conversion cartridge" which has been imple- 811
mented. Totally automatic addition of genomic data so that 812
the CDA Genomics Document is itself automatically generated 813
from the initial CDA genomics-free file, is also possible. It has 814
been demonstrated with a simple extension to this CDA 815
Genomic Document conversion cartridge, but this remains a 816
manual process at this time while the standards for insertion 817
of the document are being explored and agreed with HL7 CDA 818
committees. The current prototype merges genomic data using 819
a **gms:** namespace prefix at the end of the CDA <body>, in its 820
own CDA <section> as shown below using CDA structure as 821
an example: 822

```
<cda:clinical_document_header>
  .
  <!--header structures per CDA-->
  .
</cda:clinical_document_header>
<cda:body>
  .
  <!--clinical sections per CDA-->
  .
  <cda:section>
    <cda:caption>
      IBM Genomic Messaging System Data
    </cda:caption>
    <cda:paragraph>
      <cda:content>
        <cda:local_markup ignore="markup">
          <!--gms: tags go here-->
        </cda:local_markup>
      </cda:content>
    </cda:paragraph>
  </cda:section>
</cda:body>
```

More precisely, the cartridge looks first to see if the tags in bold 823
already exist in the document, in which case it will keep them 824
and insert there. If they are missing, it will look for a <gms: 825
body or <body tag (case-insensitively). If however there is no 826
body tag, it will insert the above before the last tag in the 827
document. 828

The result of using the CDA conversion cartridge to merge 829
clinical and genomic data is a .gms file in which all input is 830
written in its canonical GMS form. Appendix 4 shows the .gms 831
file produced by applying the cartridge to files in Appendices 832
2 and 3. The GMS commands and syntax elements have been 833
automatically generated and correctly inserted for processing 834
by the GMS encoder. 835

3.5. Output. The incoming stream is processed byte by byte 836
by the decoding GMS program and activity at the receiving 837
device is determined by the data as it arrives. This includes 838
activation of any embedded Perl applets. Passwords can be 839
automatically requested at various points to allow the stream 840
to proceed, and to unlock and read in data from any files 841
available at the receiving end (at present, additional and highly 842
compressed "background" DNA sequences). These passwords 843
are distinct from and in addition to those mentioned above. 844
Their locations in the stream are chosen by the encoding expert 845
or automatically by a legacy record conversion cartridge. 846

847 **3.5.1 Standard Outputs.** Four outputs are always generated
848 on receipt. (The default choice of root name "stream" is
849 assumed.) These represent various levels of fallback should
850 other display or conversion systems fail. Additional outputs may
851 be created by automated analysis cartridges (see below).

852 **3.5.1.1 Full Report.** The file **reports.dat** which displays and
853 interprets the entire stream with interspersed notes on any
854 errors or warnings, and with summaries of the DNA and all
855 other data transmitted. A scan of this file for occurrences of
856 the word "error" is an efficient way to use this file for debugging
857 of GMSL.

858 **3.5.1.2 XML.** The file **stream.xml** which is the stream and
859 any annotation re-expressed in XML format. This is the primary
860 file for use by downstream applications. Any CDA and even
861 XML compliance of this file is, however, subject to correct
862 preparation of the input file(s). A fast "pseudo send" and
863 "pseudo-receive" mechanism provides the preparer the op-
864 portunity to test her/his entries. This mechanism is in fact
865 always active in encoding the GMS stream – every message is
866 tested by a locally confined "sending and receiving" process.
867 Of various forms of output file, the most relevant at this time
868 is CDA XML, which can be rendered at a GUI by various
869 standard applications, such as XSL stylesheets. Hence, if the
870 original input were a CDA XML file, the output would repro-
871 duce the original CDA input save for the DNA and annotation
872 added by the expert to the original CDA clinical document.

873 **3.5.1.3 HTML.** The file **stream.html** which is an HTML
874 display generated from stream.xml. It does not require the XML
875 to be well formed. The opening tags are transformed to a
876 readable index on which the receiver can click to go to the
877 content associated with that tag. (This display is fairly general
878 to XML but some minimal assumption about CDA use is made
879 which generates still tidier output if CDA architecture is used).

880 **3.5.1.4 Default Viewer.** A basic UI, a read-only editor text
881 which is automatically invoked on receiving the GMS stream.
882 It allows interrogation of the data on the stream.xml file, and
883 follows the same indexing principle as the HTML output. It
884 does not require the XML to be well formed. Although
885 interpretation of the XML is even more basic than in the HTML
886 output, it does include a regular-expression search string facility
887 as well as standard backward-forward navigation. If it is not
888 required, entering "quit" immediately escapes the editor.

889 **3.5.1.5 Protein Sequence (Automated analysis cartridge
890 output).** A **stream.seq** file, automatically generated by the
891 protein analysis cartridge, contains the protein sequence in all
892 frame shift interpretations of the DNA, in FASTA format.

893 **3.5.2 Automated Analysis Cartridges.** Automatic cartridges
894 (replaceable code modules) are available both for processing
895 input and output in different ways. They can be thought of as
896 mini "Expert Systems" tailoring input and output for special
897 purposes.

898 An especially powerful feature of the GMS system is the
899 ability to analyze a decoded stream and augment that stream
900 with the results of the analysis. Again, GMS cartridges are used
901 on output to perform these functions. Different cartridges
902 (replaceable code modules) can be called into play depending
903 on the intended use of the final output. For example, cartridges
904 of a type which do not alter the basic information, but affect
905 display or presentation to physicians, administrators, and
906 different health experts are in development. At present there
907 is a leaning toward a research emphasis, and the cartridges
908 add automatic annotation to the DNA of the patient, translate
909 the DNA into protein sequences, and automatically annotate

the proteins, while preserving and interlacing (if required) any 910
manual annotation present in the original input. To do this, 911
two cartridge files, dna.rex and protein.rex, are currently 912
available, and a further one for diagnostic and vaccine devel- 913
opment, considering both pathogen and patient DNA, is to be 914
implemented. Appendix 5 shows the XML output after decod- 915
ing and analyzing the stream produced from the.gms file in 916
Appendix 4. Note the considerable expansion over the original 917
DNA input file in Appendix 3. 918

919 The current XML ontology used for sequence annotation is 920
a home-grown "GMS-ML" one, but is easily adapted. One 921
reason for this is that two fundamentally distinct types of 922
sequence annotation are both done at the same time. That is, 923
one is "in-sequence" (with XML tags within the DNA and 924
protein sequences), and one is "out-sequence", with external 925
tags expressing points in the sequences (e.g., residues 101– 926
105) at which a feature occurs.

4. Results 927

928 It is not the purpose of the present paper to obtain new 929
biomedical insight, or to test a medical hypothesis, but an 930
example study is useful to place the capabilities of GMS in 931
context. One interesting early proof of concept used a real 932
patient record with an arbitrary plausible DNA sequence 933
corresponding to one gene of the patient's HLA immunotype. 934
This has been extended in a related study to real patient DNA 935
data. The present example involving the data of the Appendices 936
is somewhat artificial, being modified in respect of privacy and 937
consent issues. The principle remains identical.

938 The patient data was processed as shown in Figure 1, splicing 939
the record transmitted in encrypted form from Haifa with the 940
DNA data received encrypted from IBM New York, as if from 941
the clinical laboratory. Note that unless the passwords re- 942
quested are correct, the spliced result cannot be obtained 943
because the records remain encrypted and this helps ensure 944
not only privacy, but that the correct patient receives the 945
correct data. Note also in Appendix 4 that the XML form is 946
essentially recognizable within the GMSL since whole chunks 947
of the original XML document can be retained in transmission.

948 The architecture was extended to demonstrate the modeling 949
of patient proteins as therapeutic drug targets, with structural 950
features arising from genetic features personal to each patient, 951
and which might affect drug efficacy or safety. The final return 952
of the XML to Haifa was also paralleled by the feed of the 953
translated sequence to the KRUNCH protein modeling pro- 954
gram⁴ which is accessed via a GMS cartridge through which 955
GMS generates the patient sequence in FASTA protein se- 956
quence format.

957 The sequence in FASTA format was however preprocessed 958
according to the annotation carried through from the DNA and 959
the translated sequences, in the output XML format. Any 960
polymorphism ("SNP") which destroyed for example a glyco- 961
sylation site, or signal peptide or other cleavage site, would 962
affect the corresponding automatically generated annotation 963
for corresponding glycosylation and cleavage respectively: such 964
sites would, quite correctly, not be identified, or will occur 965
elsewhere. In this sense, XML-based annotation also represents 966
modeling instructions, and in the GMSL environment this 967
concept could readily be extended to more sophisticated 968
modeling instructions. This also emphasizes the need for focus 969
not simply on functional annotation of human (and pathogen) 970
genes in medicine, but also on their post-translational and 971
maturation mechanisms.

972 An appropriate modeling template in PDB format was 973
selected being HLA-DR1 (DRA, DRB1 0101) human class II 974
histocompatibility protein (extracellular domain) complexed 975
with endogenous peptide. The protein was then modeled

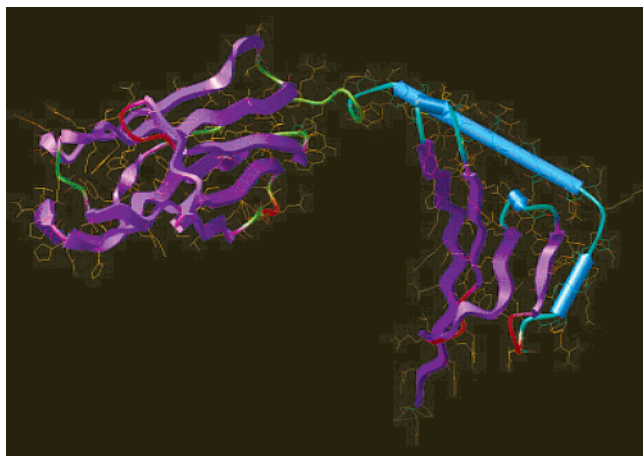


Figure 2. Example of GMS modeling of patient proteins: output of the scenario study described in the text.

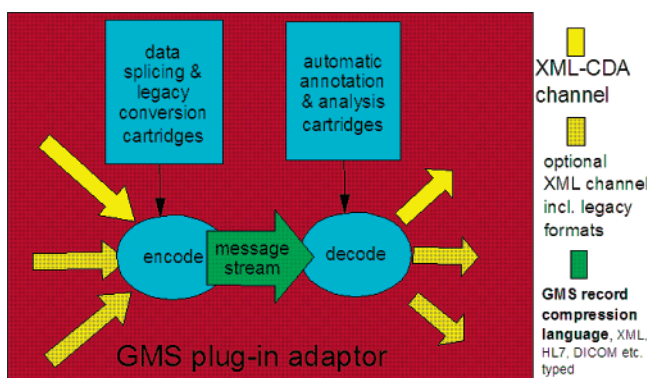


Figure 3. GMS system seen as a universal “split node” in a biological and clinical information network. The exact role depends on the plug-in cartridges.

976 against this template with the patient’s corresponding poly-
 977 morphisms (“SNPs”). The result from the initial pass is shown
 978 in Figure 2. Krunch is capable of producing reasonable
 979 preliminary models in a solvent continuum approximation,³
 980 but was then passed to molecular dynamics procedure for
 981 refinement in full solvent under periodic boundary conditions.
 982 The overall process from start of transmission from Haifa, up
 983 to the Krunch output, took approximately one minute.

984 **5. Discussion and Conclusions**

985 **5.1 Curation and Annotation of Data in GMS.** The referee
 986 raised two interesting points worth discussing. The first is that
 987 it should be emphasized as a strong feature, if GMS can indeed
 988 be extended to provide an in-line normalization and a basic
 989 data curation capability, or an interface for a plug-in to an
 990 external component that provides such functions. In fact it is
 991 the case that GMS can provide such capabilities, both by
 992 intrinsic functions (see next paragraph) and via plug-in car-
 993 tridges. In the latter case, the functionalities must be considered
 994 in terms of any larger system in which GMS fits. An early design
 995 implementation is indicated by Figures 3 and 4. Note in Figure
 996 4 that the role of the second GMS component is not only to
 997 gather and integrate patient information from the medical
 998 archives, but to link it to various analysis and annotation
 999 components: the distinction between these two roles resides
 1000 in the cartridges plugged-in in each case. PRIMA was an
 1001 important component in the early installation allowing statisti-
 1002 cal analysis and graphical display of individual and collective
 1003 records.

1004 In addition, it is relevant to recall a philosophy of GMS, i.e.,
 1005 that it can function alone in a disaster scenario. Access to

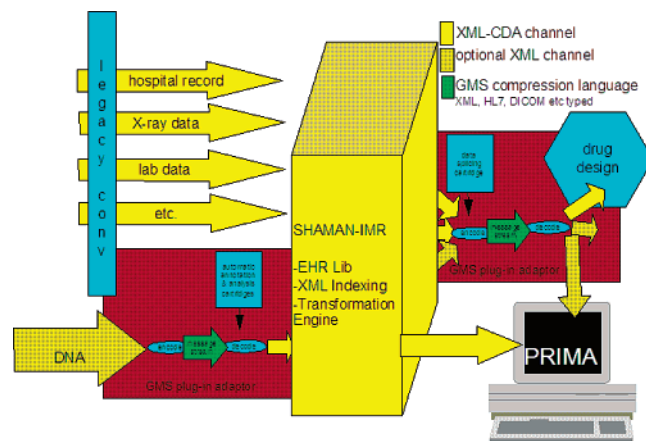


Figure 4. GMS system seen as playing two roles distinguished by the plug-in cartridges. The two roles are in this case the provision of an input channel for genomic information in parallel to other input data streams, and also the gathering and integration of the various types of patient information from the SHAMAN-IMR core archives.

1006 patient data, and in some cases access in order to further curate
 1007 or annotate that data, and in a variety of scenarios, can be
 1008 essential to patient survival or well-being. GMS can run as a
 1009 stand-alone program and some basic intrinsic functions are
 1010 always available should the overall infrastructure break down
 1011 or not be accessible, including inability to access the Internet.
 1012 As well as generating or processing XML, GMS can also directly
 1013 write an HTML document (i.e., without recourse to the XML
 1014 file) enabling domain experts to edit and correct, curate, extend,
 1015 and “sign off” on the material transmitted or stored by GMS
 1016 (for example, See Appendix 6). This HTML is available even if
 1017 the surrounding system is unavailable, including XML inter-
 1018 pretation, and can be displayed on a lap-top in isolation of
 1019 the support infrastructure (one would imagine that the patient
 1020 record or GMS files would be inputted to the lap-top on a flash
 1021 card or floppy disk, for example).

1022 The above HTML file is not the ultimate back-up to allow
 1023 access to information in the stream of GMS bytes that
 1024 represents the patient’s clinical and genomic data. GMS “close-
 1025 to-ultimate access back-up” is via a basic “command line level”
 1026 editor for inspection, searching and curation of the information
 1027 transmitted. This output is actually a transform of the informa-
 1028 tion on the XML output file into a kind of “text messaging”
 1029 format, providing a means of interpretation of the XML output,
 1030 again when further software to process the XML is not available,
 1031 but allowing more flexibility to the form of presentation than
 1032 the currently limited number of intrinsic HTML output chan-
 1033 nels (currently one per version). This flexibility is at the price
 1034 of much less elegant screen presentation, but this brings some
 1035 advantage. The output would be readily extensible, in extreme
 1036 circumstances of a disaster scenario, to allow access to patient
 1037 material on text messaging via a mobile telephone. In any
 1038 event, the availability of both this intrinsic flexible XML-
 1039 dependent “text messaging” output channel and the intrinsic
 1040 less-flexible but XML-independent HTML output channel
 1041 appears to cover most, if inevitably not all, systems failure
 1042 scenarios.

1043 Finally, there is some further support of access to (but not
 1044 easily annotation and curation of) patient information, even if
 1045 GMS itself is damaged or if there is a GMS software malfunc-
 1046 tion. The “raw” XML output itself is not difficult to read (though
 1047 it takes some familiarity). Should the stage of XML production
 1048 fail, there is also a very last resource available even beyond
 1049 those mentioned above. The stream of transmitted bytes is

Genomic Messaging System

research articles

1050 always written to a further file as a corresponding stream
 1051 relatively user-friendly mnemonics for the meaning of those
 1052 bytes. For this purpose, it is necessary that the decryption and
 1053 interpretation components retain function. GMS architecture
 1054 seeks to balance the issues of privacy and security against the
 1055 dangers of not being able to get access to patient information
 1056 when it is needed in an emergency situation. So, all of the above
 1057 access contingencies do however require that the receiver is
 1058 able to decrypt and interpret the transmitted stream of GMSL
 1059 bytes (providing he or she has the passwords). The decryption/
 1060 interpretation component is, however, a very small program
 1061 that could be easily backed-up on a floppy disk or flash card.

1062 **5.2 Institution-Dependent Privacy and Security.** The referee
 1063 also felt that it would be essential for the “scrubbing” (anonymization)
 1064 function of GMS or Clinical Laboratory Messaging
 1065 System to be ‘tailorable’ based on individual institution’s
 1066 privacy policies and IRB decisions regarding the levels and
 1067 depths of de-identifying the protected health information.

1068 The referee was correct in that some of the current security
 1069 tools in GMS are to some extent “hardwired” in the code
 1070 (though there are some choices as to the details of use,
 1071 including the extent of encryption and to whether encryption
 1072 is applied all). One reason for this “hardwiring” is that the
 1073 notion of reversibly shuffling the byte stream of GMS messages
 1074 (for privacy and security) is regarded as a fundamental benefit
 1075 of using the byte stream approach. Again, however this should
 1076 be seen as a final fall-back capability should other linked tools
 1077 fail or not be accessible. Normally, for example, standard strong
 1078 encryption can be applied even to files already encrypted by
 1079 GMS. The intrinsic GMS capability, however, is useful to retain
 1080 even when further encryption is applied. It has the benefit of
 1081 being able to specifically link encryption to the current patient
 1082 identifier so that, for example, DNA data cannot be added to a
 1083 patient record unless the identifiers match. Thus, the right DNA
 1084 is better assured to be going to the right patient record, and a
 1085 safety, as well as security, capability is enforced.

1086 In addition, it should not be forgotten that the GMS language
 1087 itself contains privacy and security commands which can be
 1088 dispersed throughout the clinical and genomic data. The details
 1089 of that dispersal would reflect the institution specific privacy
 1090 and security requirements (the overall use of these commands
 1091 can be programmed either directly as a string of use-friendly
 1092 GMSL byte mnemonics, or more typically by instructions on
 1093 the input-side plug-in cartridges). In particular, recall that
 1094 passwords can be requested at different points in the stream
 1095 giving different access to different parts of the data by approved
 1096 persons, and passwords can also be applied to give or deny
 1097 access to files which are co-transmitted or available at the
 1098 receiving site. These passwords can also be linked to the more
 1099 hardwired encryption process, and in that sense the institution-
 1100 specific security features implemented as above do confer some
 1101 installation-specific effect even on the way that the “hard-
 1102 wired” component is ultimately effective. There is a restricting
 1103 preference that a password request command is the first line
 1104 on a .gms file and that the first data statement string must
 1105 somewhere contain an encrypted patient identifier. However,
 1106 it may be also be automatically substituted in the data
 1107 statement by the variable {identifier} represented in the byte
 1108 stream, again allowing some flexibility as to how the system is
 1109 set up for a specific institution.

1110 **5.3 Overview.** Taking account these above observations (5.1,
 1111 5.2), the following may be stated in summary. The Genomic
 1112 Messaging system (red box) is also a “smart” plug-in multi-
 1113 branched adaptor at any juncture in a clinical or biological IT
 1114 network where genomic data is at least partly involved. The
 1115 exact role and branching at any point is determined by choice
 1116 of cartridges which convert legacy formats, and write GMS

language (GMSL) controlling use, compression, privacy issues 1117
 of encryption and access to files, and checking. It allows 1118
 queuing, and disaster recovery in view of the fact that it splits 1119
 each node into a file write-send-read process where data can 1120
 be given safe harbor. Importantly, it links medical records to 1121
 pure biological research tools. At present this includes those 1122
 of IBM’s Computational Biology Center, and other installations 1123
 capable of reading XML, spreadsheets/CSV files, and bio- 1124
 sequence and bio-structure data in FASTA and PDB protein 1125
 formats, respectively. 1126

1127 **5.4 Protein Modeling Study.** The purpose of the protein
 1128 modeling study was not an abstract study in protein science,
 1129 but to explore the application of bioinformatics in personalized
 1130 medicine, involving direct use of real patient records. In
 1131 addition, the epitope data passed with the patient record
 1132 (Appendix 5) was used to form the basis of an endogenous
 1133 binding peptide with D-amino acids. The aim was one of
 1134 designing a proteolytic peptidomimetic inhibitor in the hope
 1135 that it might favorably regulate bone marrow transplant
 1136 rejection by that patient, given the available donor material.
 1137 Though the design process was interactive, automatic methods
 1138 have been developed⁵⁻¹¹ in the previous laboratory of one of
 1139 the authors (BR), and it is clear that in certain cases, an
 1140 automated design process is possible (the automatic drug
 1141 modeling code was sold to a specific biopharmaceutical
 1142 company and was not accessible to the authors at the time of
 1143 the study). 1144

1145 The study was to demonstrate feasibility in principle of
 1146 personalized medicine including not just selection, but poten-
 1147 tially de novo design, of chemical agents specifically for the
 1148 patient. Interest in HLA relates to the long understood high
 1149 degree of polymorphism between HLA genes in different
 1150 individuals, and the method of D-peptidomimetic design and
 1151 synthesis could build on recent advances in relevant tech-
 1152 nologies.¹²⁻¹⁷ Thus, this kind of application might be expected
 1153 to provide one of the earliest proofs of truly information-based,
 1154 fully automated, personalized molecular medicine. 1155

1156 That said, it is important to note that the proof of concept
 1157 in the present paper was solely from the information technol-
 1158 ogy perspective. It was not to assess the accuracy of any one
 1159 modeling method, nor to propose a particular biological
 1160 rational for a therapeutic, and certainly not to propose a design
 1161 which is to be validated for pharmaceutical efficacy. Clearly,
 1162 even the aspiration for such would be premature: much needs
 1163 to be done before such procedures are feasible, in accord with
 1164 future legislation, and routine. However, there is no reason to
 1165 believe that a profoundly different information technology is
 1166 required at the GMS level of contribution. Rather, the develop-
 1167 ments will be at the molecule modeling and pharmaceutical
 1168 design level. In the interim, the GMS is a pilot example of a
 1169 system which may help clinical and other biomedical scientists
 1170 perform research to achieve such goals. 1171

Appendix 1 1169

1170 **GMS Language Commands.** A .gms file comprises a field of
 1171 DNA base pair characters AGCT in which commands are
 1172 embedded. Some commands called **data commands** are fol-
 1173 lowed by data, and delimiters which indicated where that data
 1174 begins and ends. They isolate important bioinformatic and
 1175 clinical annotation from the surrounding filed. Otherwise, DNA
 1176 rules!: all data is assumed to be DNA or commands related to
 1177 DNA stream management, delimited separated by semicolon
 1178 ‘;’ or newline (\n, linefeed, carriage return). Spaces (whitespac-
 1179 es) are ignored although if commands consist of multiple
 1180 words, at least one must occur between the words. 1180

1181 Recall below that the GMS language script will be generated
 1182 on a .gms input file which is the final input to GMS. The default
 1183 root name for this and other files is assumed to be stream 1183

research articles

Robson and Mushlin

1184 throughout. Hence the .gms file is stream.gms. The major
 1185 stream information is then stream.gmb: this the binary stream
 1186 or file which can be encrypted and/or further compressed, by
 1187 the gMS program. In what follows, all data is written to
 1188 stream.gms, and some (specified by index, comment, and
 1189 bracket commands, will also be written to stream.gmb, the
 1190 binary messaging stream. File stream.xml is generated on
 1191 receipt and decoding (however an equivalent file is also
 1192 generated on encoding and is compared as a check prior to
 1193 sending).

Most used example phrases and recipes

password:{{identifier}}{by my secret word protect data}}	Preferred first line on .gms file. For security the first data statement string must somewhere contain an encrypted identifier. It may be automatically substituted in the data statement by {identifier}. The first data statement will typically be a password data statement and ask for password too, checking it against the "password" specified, here the string "my secret word".
data: Main security check - password required\ password: Patient # 1 {by shaman protect data}}	Use of data statement as message in subsequent passwords.. "Main security check - password required" is shown at screen when program halts for password.
data: <gene ID=LA:HLA00664 801 bp's, transplantation example> index;	Writing a user-provided xml tag to stream.xml.
xml {<my_tag feature="asian polymorphism" locus="{locus}", sequence="{sequence}"/> }; index:and protein	Writing an xml tag to stream.xml with standard GMS variables, inserting it into the DNA sequence at that point and at the corresponding point in the protein translation.
data: testing testing{employ data} xml: <test>{data}</test> end of data} index	Define the multipurpose GMS variable {data} as the string "testing testing" to write it in xml tags <test> to stream.xml (can be reused)
name: my_tag ; xml: <scrubbed_data example="1"/> define data}	Defining xml tags or other data for reuse, here stored in a variable called "my_tag"
name: my_tag ; xml: {recall data} name: mary ; xml: {elizabeth define data}	Define the user variable {mary} as the string "elizabeth" for future use in data statements.
xml: <test> patient {identifier} has informal code name {mary}</test> ;index	Invoke GMS variable containing patient identifier. Write to stream.gmb and stream.xml
hl7 <hl7 bracket="{bracket}" level="{level}"> MSH ^~& XRAY CDB ORU^R01 K172 P<er> PID PATID1234^5^M11 JONES^WILLIAM 196106 13 M<er> OBR P8754^OE XR1501^XR 71020^Chest X-ray PA 198703281530 198703290800<er> OBX 1 TX 71020 It is a normal PA Chest X-ray F<er </hl7> end of data} ;index	Writing hl7 in an XML context to stream.xml. Example use of variables, here {bracket} {level} reporting current status of ')' and '(' bracketing commands (see below).
dicom: dfufhh7754D 432456F543%..... end of very long terminator }	Send image data in stream.gmb with terminator defined at start of use of GMS, unlikely to be encountered in long compressed streams (a priori probability is estimated).
perl ; print "Perl applet test OK\n";print "Hit return:"; <STDBN>: {applet data}	Sending a Perl applet in the stream.gmb. Activates when encountered.
A C G ; G G ; T ; deletion; snp G AAAAGGCGCGTtagCCCC;	Place stream of DNA in stream.gmb and stream.xml. Put ACG in one byte, GG in one byte, T in one byte, record snp deletion at next point, record snp G at ne_xt point, pack rest as much as possible three per byte.
filedata: {by shaman unlock data} number: 15 base pairs squeeze dna * AGCTTCAGAGCTGCT	Compressing DNA four base pairs per byte (one base pair per two bits), and writing it to stream.gmb and stream.xml. Password, here "shaman", must have been entered by last used password command.
filedata: template1.gms by my secret word unlock data} ; read in dna	As above but read DNA (with * and) from file specified (template1.gms) on receipt. Padlock file. Password, here "my secret word" must have been entered by an last used password command.
01001011;	Research version only: write the byte corresponding to this binary

Commands which insert DNA into current sequence

singlet X	Place X (AGC or T) as next current sequence and store 1 per byte in stream.gmb
X;	Place X (AGC or T) as next current sequence and store 1 per byte in stream.gmb
doublet XX	Place XX (AGC or T) as next current sequence and store 2 per byte in stream.gmb.
XX	Place XX (AGC or T) as next current sequence and store 2 per byte in stream.gmb.
triplet XXX	Place XXX (AGC or T) as next current sequence and store 3 per byte in stream.gmb.
XXX	Place XXX (AGC or T) as next current sequence and store 3 per byte in stream.gmb.
snp X	Place X (AGC or T) as next current sequence and record as a single nucleotide polymorphism, and store as 1 per byte in stream.gmb.
XXXXXXXXXXXXXX	Pack as much as possible into triplets (i.e. 3 per byte) in stream.gmb. Stragglers modulo 3 are put into singlets or doublets.

General commands (no arguments; DNA annotation and validation checks)

insertion (Place marker in stream.gmb that DNA begins here..
) insertion	Place marker in stream.gmb that DNA insertion end here
deletion	place marker in stream that DNA deletion occurs here
end of task	Terminate whole task. Ignore following information.
index	Write content of last data statement (usually an xml data statement) on stream.xml.
and protein	Qualify last used index statement to write content of last data statement on stream.xml and reproduce at same location in the translated protein sequences reported on stream.xml. Viz:- index; and protein;
(escape character)	Required for internal working of GMS. Also, ensure data termination if ' content of a preceding data statement abnormally skipped.
]	Perform parity check on last data statement Also the optional post-terminator symbol, see below.
[No action in Version 00. Useful in conjunction with '[' to bracket sets of commands and data in the stream. Also an initiator symbol (see below) if follows data command such as data, viz:- data:
danger	Flag a warning. 00000000 coding this command should not be encountered in normal mode.
validate	Advance validation counter 1

Commands which write xml (to stream.xml)

comment	Write to stream.xml: <![CDATA[<gms:comment type="{bracket}" state="{state}" type="{type}" level="{level}"> <gms:comment> > contents of previous data statement
new dna	Write to stream.xml <gms:automated_annotation> <gms:dna sequence="{sequence} base="{base}" locus="{locus}"> Here: all annotation tags added manually plus all added automatically, including automatic translation and protein annotation. <gms:dna> <gms:dna sequence="{sequence} base="{base}" locus="{locus}">. Used first time, sets sequence count to 1. If used again, advances the sequence counter to indicate new sequence and writes
end of dna	Add special end null entry, and properly close 'new dna' annotation tags on stream.xml. {end}</gms:dna> <gms:dna_checks> summary data </gms:dna_checks> <gms:automated_annotation>
{n e.g. (63	Write a self-standing xml annotation tag to stream.xml <gms:open feature="{text}" type="{n}" level="{level}"> where text is the content the last data statement, n is the bracketing type n, and level is the bracketing level for this type. n may be 0..63
{n e.g.}63	Write a self-standing xml annotation tag to stream.xml <gms:close feature="{text}" type="{n}" level="{level}"> where text is the content of the last data statement, n the bracketing type n, and level is the bracketing level for this type. n may be 0..63

A data statement comprises the data command followed by initiator symbol or symbols, the stream of data which will be encoded one byte per character, and the terminator symbol or symbols. The data command format is

Data_command, delimiter, initiator, ...data..., terminator-[optional post terminator command], where the semicolon or newline character is required as the usual delimiter for commands (and commas above are for clarity and do not appear in actual use).

Genomic Messaging System

research articles

1203 Note that, as with the delimitation of all commands, the
 1204 semicolon can be omitted and replaced by the newline
 1205 character (“\n”, “carriage return”, “linefeed”), and interspersed
 1206 whitespace is not significant. An example appearance of a data
 1207 statement is:-

data

`\my list of data\`

1208 For security, the first data statement, but not necessarily the
 1209 first information, on the.gmi file, has a special status. It is
 1210 typically a password statement which is a special case of a data
 1211 statement as described below. Comment may be added around
 1212 {identifier} and within the enclosing square brackets, but not
 1213 after the {by my_password protect data} statement. There are
 1214 two implied passwords – that implied by {identifier} and that
 1215 implied by my_password. “my_password” is any other word or
 1216 phrase (a word like Mary would do equally well) selected by
 1217 the preparer of the .gmi file, and must be entered by
 1218 the receiver on receipt when requested. The password “{identifier}”
 1219 must be satisfied even before the decoded stream gets as far
 1220 as asking for the above password. The string as written in the
 1221 .gmi file is either exactly that string “{identifier}” of 2 curly
 1222 brackets and 10 alphabetic letters, or alternatively, a string
 1223 identical to that generated by the encrypted form of “Enter
 1224 Identifier” request above, e.g., 01gC/VymItaB. In effect {identi-
 1225 fier} is the variable which automatically inserts the string e.g.
 1226 01gC/VymItaB into the password statement.

1227 The command *data* in the above example is an example of
 1228 a specific data command among several other possible specific
 1229 data commands, and they collectively represent generic or
 1230 general data commands. These other specific commands are
 1231 as follows. Those in brackets below perform no further sup-
 1232 ported function in version 00 but are acceptable as general data.

Data commands (arguments following in [...data] or sometimes \.,\, or [...])

Data command	Assumed data content	Special action	Termination. Use \ or escape character as data terminator or may \ appear as content?
data	Any data.	None.	\ terminator or (end of data). If terminated by (employ data), this data will substitute the string (data) in any subsequent data statement. If terminated by (store data), it will substitute for (name) wh. are name is the contents of previous data statement.
perl	Executable Perl	If terminated by {applet data}, automatically runs applet.	Usually {applet data}. Not executed on (end of data)
dna	DNA data to be stored 'classically' as the ASCII characters AGCT including new characters	None, except to prevent this data from being inserted into current dna sequence. Instead it can be stored in a variable name using (store data) (see 'data' above)	\ terminator or (end of data)
squeeze dna	DNA to compress 2 bits per bp, viz:- number:{5 base pairs};squeeze dna; * AGCTC \	Add to current dna sequence and squeezes into 4 bp's per byte as much as possible. Stragglers left modulo 4 are stored as singlet, doublet or triplet commands	Termination controlled by number in previous data statement but \ should be present as check.
read in dna	As squeeze dna, but gets data from file, viz:- filedata;[template1.gms[by shaman unlock data]]	As squeeze dna.	As squeeze dna, but end of file condition terminates too.
number	Any integer embedded in text up to decimal point.	Data ignored (deleted) after decimal point '.'.	\ terminator or (end of data). (store data) and (employ data) also useful.
xml	Any valid part of XML compliant document	None	\ terminator or (end of data). (store data) and (employ data) also useful.

hl7	Any HL7	None	Content only. use (end of data)
dicom	DICOM data	None	Content only. use (end of data). For large amounts of data, red_efine end-of-data at startup (this calculates for the user the probability of chance occurrence.
base pairs	As command dna, but normally used to add annotation referring to short segments	None, except to distinguish it from command 'dna' above and also from dna to be inserted into current dna sequence.	\ terminator or (end of data). (store data) and (employ data) also useful.
protein	Protein data to be stored 'classically' as the ASCII characters GAVLIST... Including new characters	None	\ terminator or (end of data). (store data) and (employ data) also useful.
filedata	File name or other file instructions, e.g., filedata;[template1.gms [by password unlock data]];read in data	Report possible need for password, in run record. If text data is present, defines active external file as other than template.gmb. If (by password unlock data) is the data terminator, the last entered password is required to continue.	Usually (on password unlock data). \ terminator valid if no password check required to continue.
password	Any text. Not used as password, but text may be comment or used later as it would be by any data statement	Halts screen and request password. If data terminates with (on password protect data) compare string password with entry, otherwise keep till it check requested.	Usually (on password protect data). \ terminator is valid and password becomes last password entered, but check is not done at that time.
instruction	Reserved for extended instruction set	Switch instruction mode.	Usually \ terminator, or (end of data)
name	General name for something	None	Usually \ terminator, or (end of data)
xml cdata	As xml but preferred for use for cdata intensive data	None	Usually \ terminator, or (end of data)
conditional	Conditional action, not implemented	Perform condition/logic test	Usually \ terminator, or (end of data)
skip	Number of bytes to skip if number, to or label to skip to if last condition tested is true. Not implemented.	Perform skip.	Usually \ terminator, or (end of data)
label	Data is label for reference.	None	\ terminator or (end of data). (store data) and (employ data) also useful.

1233 In the example data;\my list of data\, the first backslash
 1234 represents the initiator and the second the terminator. Possible
 1235 initiators are:

Initiator commands

\	Read data till terminator or end of file is encountered.
[Read data till terminator or end of file is encountered. Usually used for style in conjunction with character] described below.
*n	Read following n characters where n is 1-255 or end of file is encountered. e.g *100. Usually first non-comment "# line when data is on background file.
*	Read following N characters where N is the numeric value in the preceding data statement or end of file is encountered. Usually first non-comment "# line when data is on background file, in which case the preceding data statement is in the main .gmb file.

Possible terminators are the following.

1236

Terminator commands

escape character (appears as backarrow on screen)	Terminate data except for perl, hl7, dicom data types (which may use this character).
end of file	Terminates any data stream being read on main or background file.
\ (i.e. backslash)	On reading GMS file, replace by escape character as above.
{end of data}	Terminate data of any type. This and other 'data' commands (note whitespace) below collectively provide less than a one in a 256 ¹² a priori probability of chance encounter of characters which will inadvertently prematurely terminate a stream.

research articles

Robson and Mushlin

{end of terminatorstring}	When specified at startup of encoding, all terminator commands ending 'data}' (note whitespace) are inactivated and replaced by corresponding terminatorstring commands
{applet data}	Terminate data of any type and immediately attempt to execute data assuming it to be Perl script.
{define data}	Terminate data of any type and store under the name name defined by the data in the preceding data statement. Also, create general substitution variable {name} storing that data.
{recall data}	Terminate data of any type and replace all of that data, if any, by data stored under the name defined by the preceding data statement.
{employ data}	Terminate data of any type and store data as a string in the specific internal variable called KEEPDATA.
{deploy data}	Terminate data of any type and replace any and all occurrences of string '{data}' in that data by the string stored in the specific internal variable called KEEPDATA.
{by password protect data}	Terminate data of any type and abort if current active password does not match password specified.
{by password unlock data}	Terminate data of any type and abort if current active password does not match password specified. If it does match, the data is interpreted as the name of an external file to be opened and the suffix .gmb is added. If the suffix .gmb or .gms is already represented in the filename, suffix .gmb is assumed. If the data is empty the standard root filename is assumed.

1237 If the string 'data}' is encountered and the terminator is not
 1238 one of the above terminator commands ending in 'data}', the
 1239 error "unidentified terminator qualifier construct" is reported
 1240 and GMS decoding aborts.

Optional post-terminator symbols

	Perform parity check on data. In effect, calculate 0 or 1 to send in stream as below.
0	Parity must be even, abort otherwise
1	Parity must be odd, abort otherwise

1241 Substitution variable commands can appear anywhere in a data
 1242 statement. They are replaced by a current value of a corre-
 1243 sponding variable in the GMS program.

Substitution variable commands

{version}	Current version of GMS system used to encode. If this variable is used, it must match decode version used or GMS aborts on decoding.
{identifier}	Current patient identifier entered in startup of encoding, or, if the "scrub" mode is set at that time, its encrypted form. The string '{identifier}' can be used throughout the initial .gmi input file (not just in data statements) and is assigned only on initiation of encoding. If it is not matched on decoding, GMS decoding aborts.
{id}	Current patient identifier encrypted or not according to whether the scrub mode is set or not.
{open id}	The unscrubbed identifier
{closed id}	The scrubbed identifier
{scrub status}	1 is scrubbing is set, 0 otherwise
{density}	Very roughly, chances of recognizing just one word after encryption has taken place.
{point}	Current command byte being decoded. Useful in indexing to the raw GMS binary stream.
{xml symbols}...{end xml symbols}	Replace all intervening non-xml symbols by their standard XML protected & 'ampersand' forms.
...{treat as peptide}	Convert all preceding part of data to formally correct definition of peptide or protein sequence, and add automatic annotation (if cartridge is present).
{sequence}	Current DNA sequence number
{locus}	Current base pair number in that sequence
{base}	Current accumulative base pair count over all sequences (i.e., not reset per sequence).

{data}	Current string stored in internal KEEPDATA variable
{sourcefile}	Current main sourcefile root name (default 'stream')
{backgroundfile}	Current background or 'template' file root name specified by filedata command
{bracket}	The last type of GMS bracket used e.g. 2
{level}	The bracket level of the last type of GMS bracket used e.g. 2
{state}	The state 'open' or 'closed' of the last type of GMS bracket used.
{name}	The string stored in the variable name defined by any earlier {define data} terminator

Appendix 2

1244

1245 **Sample Extract CDA .gmi File.** This is the patient record
 1246 data feed. This is an extensive document; parts have been
 1247 removed both to meet privacy and consent issues as well as
 1248 for brevity.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XML Spy v4.0.1 U (http://www.xmlspy.com)-->
<leveltwo xmlns:gms="GMS_schemas" xmlns:bmt="BMT_schemas" xmlns:cda="KAI_CDA"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="BMT_schemas BMT_Discharge.xsd">
  <cda:clinical_document_header>
    <cda:id EX="BMT_1630" />
  <!--Based on the spec "LOINC_Document_Names_Spreadsheet.xls" from LOINC-->
  <cda:document_type_cd S="2.16.840.1.113883.6.1" DN="SE.Bone Marrow" />
  <cda:origination_dctm V="2001-09-02" />

  *
  *
  (Content removed for confidentiality)
  *
  *

  <Instructions_To_Patient_And_Family>
    <cda:body>
      <cda:section />
    </cda:body>
  </Instructions_To_Patient_And_Family>
</At_Discharge>
</BMT_Discharge_Body>
<cda:section>
  <cda:caption>IBM Genomic Messaging System Data</cda:caption>
  <cda:paragraph>
    <cda:content>
      <cda:local_markup ignore="markup">
        </cda:local_markup>
      </cda:content>
    </cda:paragraph>
  </cda:section>
</leveltwo>
```

Appendix 3

1249

Sample DNA .gmd File.

1250

```
<gms:genomic_data>
<gms:gene sequence="HLA00664,DRB1*0101,801 bases, 80C9FCB6 checksum" />

<gms:t_cell_epitopes>
  <gms:protein>pkyvqkntlkl</gms:protein>

  <gms:protein>gplkaeiaqrle</gms:protein>
</gms:t_cell_epitopes>

ATGGTGTGTC TGAAGCTCCC TGGAGGCTCC TGCAATGACAG CGCTGACAGT GACACTGATG GTGCTGAGCT
CCCCACTGGC TTGGCTGGG
<gms:experimental_start_of_mature_peptide />

GAC ACCCGAC CAGCTTCTT GTGGCAGCTT AAGTTTGAAT GTCATTTCTT CAATGGGAGC GAGCGGGTGC
GGTTGCTGGA AAGATGCATC TATAACCAAG AGGAGTCCGT GCGCTTCGAC AGCGACGTG
<gms:snp>G GG
<gms:annotation>allotype is g in drb1*0101, r in drb1*0105</gms:annotation>
</gms:snp>

GAGTACCGGGCGGTGACG GAGCTGGGGC GGCCGTGATGC CGAGTACTGC AACAGCCAGA AGGACCTCCT
GGAGCAGAGG CGGGCCGCGG TGGACACCTA CTGCAGACAC AACTACGGGG TTGGTGAGAG CTTACAGATG
CAGCGCGGAG TTGAGCTTAA GGTGACTGTG TATCCTTCAA AGACCCAGCC CCTGCAGCAC CACAACCTCC
TGTTCTGCTC TGTGAGTGGT TTCTATCCAG GCAGCATGA AGTCAGGTGG TTCGGGACG CCGCAGGAAG
GAAGGCTGGG GTGGTGTCCA CAGGCTGTAT CAGAATGGA GATTGGACCT TCCAGACCCCT GGTGATGCTG
GAACAGATTC CTGGAGTGGG AGAGGTTTAC ACCTGCCAAG TGGAGCACC AAGTGTGACG AGCCCTCTCA
CAGTGGAAATG GAGAGCACGG TCTGAATCTG CACAGGCAAG GATGCTGAGT GGAGTCGGGG GCTTCGTGCT
GGCCCTGCTC TTCCTTGGGG CCGGGCTGTT CATCTACTTC AGGAATCAGA AAGGACACTC TGGACTTCAG
CCAACAGGAT TOCTGAGCTG A
<gms:new_fragment>
<gms:annotation>patient polymorphism</gms:annotation>

AGGAATCAGA AAGGACACTC TGGACTTCAG CCAACAGGAT ACCTGAGCTG A</gms:new_fragment>
</gms:genomic_data>
```

Appendix 4

1251

1252 **Sample .gms Extract of File from CDA Cartridge (header**
 1253 **and clinical sections removed).**

Genomic Messaging System

research articles

```

*
*
* (leading GMSL -in relation to password including patient identifier- deleted)
*
*
xml;{
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XML Spy v4.0.1 U (http://www.xmlspy.com)-->
<leveltwo xmlns:gms="GMS_schemas" xmlns:bmt="BMT_schemas" xmlns:cda="KAI_CDA"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="BMT_schemas BMT_Discharge.xsd">
<cda:clinical_document_header>
</cda:clinical_document_header>

<cda:section>
<cda:caption>IBM Genomic Messaging System Data</cda:caption>

<cda:paragraph>
<cda:content>
<cda:local_markup ignore="markup">{end of data}};index; xml;{
<gms:annotation>{xml symbols} GMS-augmented document created Sat
Dec 1 17:25:59 2001 gms:environment tags allow use of valid xml as annotation
mixed with DNA in ..GATTACCA.. format, and executable GMSL (Genomic Messaging
Stream Language) as content. The GMSL will activate immediately when program
gms is run with the IBM-Yorktown legacy conversion cartridge option selected
for IBM-Haifa CDA hospital files. {end xml symbols}</gms:annotation>

{end of data}};index; xml;{
<gms:genomic_data>{end of data}};index;new dna xml;{
<gms:gene sequence="HLA00664,DRB1*0101,801 bases, 80C9FCB6
checksum" />

{end of data}};index;and protein xml;{
<gms:t_cell_epitopes>{end of data}};index;and protein xml;{
<gms:protein>{end of data}};index;and protein;protein;[
pkyykqntklka {treat as peptide}\};index;and protein;xml;[</gms:protein>

{end of data}};index;and protein; xml;[
<gms:protein>{end of data}};index;and protein;protein;[
gplkaeiagrle {treat as peptide}\};index;and protein;xml;[</gms:protein>

{end of data}};index;and protein; xml;[</gms:t_cell_epitopes>
TGCTGATGACAG
CGCTGACAGT GACTGATG GTGCTGAGCT CCCACTGGC TTGGCTGGG; xml;{
<gms:experimental_start_of_mature_peptide />

{end of data}};index;and protein GAC ACCCGAC CACGTTTCTT GTGGCAGCT
AAGTTTGAAT GTCTTTCTT CAGTGGAGC GAGCGGGTGC GTTGCTGGA AAGATGCATC TATAACCAAG
AGGAGTCCGT GCGCTTCGAC AGCGACTG xml;{
<gms:snp>{end of data}};index;and protein G GG xml;{
<gms:annotation>{xml symbols} allotype is g in drb1*0101, r in
drb1*0105 {end xml symbols}</gms:annotation>

{end of data}};index; xml;[</gms:snp>

{end of data}};index;and protein GAGTACCGGCGGTGACG GAGCTGGGG
GGCCTGATGC CGAGTACTGG AACAGCCAGA AGGACTCTCT GGAGCAGAGG CCGGCGCGG TGGACACCTA
CTGCAGACAC AACTACGGGG TTGGTGAGAG CTTCACAGTG CAGCGCGGAG TTGAGCTTAA GTTACTGTG
TATCCTTCAA AGACCCAGCC CCTGCAGAC CACAACCTCC TGGTCTGCTC TGTGAGTGGT TTCTATCCAG
CAGCATTTGA GATCAGGTGG TTCGGAACGC GCCAGGAAGA GAAGGCTGGG GTGCTGTCCA CAGGCTTGA
CCAGATTGA AGTSGACCT TCACAGCCCT GGTGATGCTG GAAACAGTTC CTGSGASTG AGAGCTTTAC
ACTCCGAG TGCTGACCTC AAGTGTGAGC AGCTGCTCA CAGTGAATG GAGGACAGC TCTGAAATGC
CAGAGAGCA GATCCTGAGT GAGTGGGG CTCTGTGCTT GGGCTGCTC TTCTGTGGG CCGGCTGTT
CATCTACTTC AGAATCAGA AAGGACACTC TGGACTTCAG CCAACAGGAT TCTGTAGCTG A new dna
xml;{
<gms:annotation>{xml symbols} patient polymorphism {end xml
symbols}</gms:annotation>

{end of data}};index; AGGAATCAGA AAGGACACTC TGGACTTCAG CCAACAGGAT
ACCTGAGCTG A end of dna xml;[</gms:genomic_data>

\};index; xml;[</cda:local_markup>
</cda:content>
</cda:paragraph>
</cda:section>
</leveltwo>
*
*
* (further GMSL deleted)
*
*

```

```

la
<gms:protein_feature type="whole_sequence" sequence="1"
context="binding peptide?" readingframe="3 complement" start="1" stop="12" />

<gms:protein_feature type="pkc_phosphorylation"
sequence="1" readingframe="3 complement" start="8" stop="10" />
</gms:protein>

<gms:protein>gplkaeiagrle
<gms:protein_feature type="whole_sequence" sequence="1"
context="binding peptide?" readingframe="3 complement" start="1" stop="12" />
</gms:protein>
</gms:t_cell_epitopes>

ATGGTGTGTCTGAAGCTCCCTGGAGGCTCCTGCATGACAGCGCTGACAGTGCATGATGGTGTGCTGAGCTCCCACTG
GCTTTGGCTGGG

<gms:experimental_start_of_mature_peptide />

GACACCCGACCAAGCTTTCTTGGCAGCTTAAGTTTGAATGTCATTTCTCAATGGGACGGAGCGGGTTCGGTTGCTG
GAAAGATGCATCTATAACCAAGAGGAGTCCGTGCGCTTCGACAGCGAGCTG

<gms:snp>GGG
<gms:annotation>allotype is g in drb1*0101, r in
drb1*0105</gms:annotation>
</gms:snp>

GAGTACCGGGCGGTGACGAGCTGGGGCGGCTGATGCGGACTCTGGAACAGCCAGAAGGACTCTCTGGAGCAGAGG
CGGGCCGGGTGGACACCTACTGCAGACACCACTACCGGGTTGGTGAGAGCTTCAAGTGCAGCGGAGTTGAGCGCT
AAGTGTACTGTGTACTCTCAAAGCCCGCCCTCAGCACCACCACTCTGGTGTGCTGTGAGTGGTTCAT
CCAGGACGATTAAGTACAGTGGTTCGGAACGCGCCAGGAAGAGAGGCTGGGTTGTCTCCACAGCGGATCCAG
AATGGAGATTGACCTTCAGACCTCGTGTGCTGAAACAGTTCCTCGGAGTGGAGAGGTTTCACTCCCAAGTG
GAGCACCCAGTGTGACGAGCCCTCTCAGTGGAAAGAGCAGCTGCTGAATCTGCACAGCAAGATGCTGAGT
GGATGGGGGCTTGTGCTGGGCTGCTCTTCTTGGGGCCGGCTGTCATCTACTCAGGAATCAGAAAGGACCA
TCTGGACTTCAGCCAAAGGATTCCTGAGCTG

<gms:automated_annotation>
<gms:length sequence="1">0</gms:length>

<gms:bases_so_far sequence="1">0</gms:bases_so_far>

<gms:invalid_dna_symbols>
</gms:invalid_dna_symbols>

<gms:agct_count>a171 g263 c200 t167</gms:agct_count>

<gms:agct_ratio>a21% g33% c25% t21%</gms:agct_ratio>

<gms:gene sequence="HLA664,DRB1*11,81 bases, 8C9FCB6
checksum" />

<gms:t_cell_epitopes>
<gms:protein>pkyykqntklka {treat as
peptide}</gms:protein>

<gms:protein>gplkaeiagrle {treat as
peptide}</gms:protein>
</gms:t_cell_epitopes>

<gms:orf>ATGGTGTGTCTGAAGCTCCCTGGAGGCTCCTGCATGACAGCGCTGACAGTGCATGATGGTGTGCTGAGC
TCCCACTGGCTTGGCTGGG

<gms:experimental_start_of_mature_peptide />

GACACCCGACCAAGCTTTCTTGGCAGCTTAAGTTTGAATGTCATTTCTCAATGGGACGGAGCGGGTTCGGTTGCTG
GAAAGATGCATCTATAACCAAGAGGAGTCCGTGCGCTTCGACAGCGAGCTG

<gms:snp>GGG</gms:snp>

GAGTACC

<gms:transcription_factor>GGGCGG</gms:transcription_factor>

TGACGGAGCTG

<gms:transcription_factor>GGGCGG</gms:transcription_factor>

CCTGATGCCGAGTACTGGAACAGCCAGAAGGACTCCTGGAGCAGAGGGCGGCTGGACACCTACTGCAGACAC
AAGTACCGGGTTGGTGAAGACTCAGTGCAGCGGAGTGGAGCTTAAGTGCATGCTGTGTATCCTTCAAAGCCAG
CCCTGCAGCACCAACCTCTGGTGTCTGTGAGTGGTTCATCCAGGACAGTGAAGTCAAGTGGTGTCCGG
AACCGCCAGGAAGAAGGCTGGGTTGGTGTCCAGGCTGATCCAGAATGGAGATTGCAAGTCCAGACCTGGTG
ATGCTGGAACAGTCTCGGAGTGGAGGTTTACAGCTGCAAGTGGAGCACCAAGTGTGACGAGCCCTCACA
GTGGAATGGAAGGACCGTCTGAATCTGCAGAGCAAGATGCTGAGTGGAGTCCGGGGCTCTGTGCTGGGCTGCT
TTCTTGGGGCCGGCTGTTCACTACTTCAAGGAATCAGAAAGGACACTCTGGACTTCAGCCAAAGGATTCCTGAGC
TGA</gms:orf>

<gms:dna_feature type="whole_sequence" sequence="1"
start="1" stop="801" />

<gms:dna_feature type="gene sequence=HLA664,DRB1*11,81
bases, 8C9FCB6 checksum/" sequence="1" start="1" />

<gms:dna_feature type="t_cell_epitopes" sequence="1"
start="0" stop="0" />

<gms:dna_feature type="protein" sequence="1" start="0"
stop="0" />

<gms:dna_feature type="protein" sequence="1" start="0"
stop="0" />

<gms:dna_feature type="orf" sequence="1" start="1"
stop="801" />

<gms:dna_feature
type="experimental_start_of_mature_peptide/" sequence="1" start="91" />

<gms:dna_feature type="snp" sequence="1" start="220"
stop="222" />

<gms:dna_feature type="transcription_factor" sequence="1"
start="230" stop="235" />

<gms:dna_feature type="transcription_factor" sequence="1"
start="247" stop="252" />

<gms:protein sequence="1" readingframe="1"
length="267">MVCLKLPGGSCMTALVTLMVLSPLALAGTRPRFLWQLKFECHFNGT
RVRLLERICYINQESVRFSDVGEYRAVTELRPDAEYWNQKDLLEQR
AVDVTCHRYNGVGSFTVQRVVEPKVTVPYPSKQPLQHNLVCSVSGFY
GSIEVRNFRNGQBEKAGVVSGLIQNGDWTPTQLVMEVTPRSGEVYTC
EHPVSPTPLTVEWRASESAQSKLSGVGGPVLGLLPLAGALFIYFRNQK
HSLQLQTPGFLS.</gms:protein_sequence>

<gms:protein_annotation>
<gms:translation sequence="1" readingframe="1">
<gms:gene sequence="HLA00664,DRB1*0101,801 bases,

```

1254 Appendix 5

1255 Sample .xml File after Automatic Annotation (header and
1256 clinical sections removed).

```

<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XML Spy v4.0.1 U (http://www.xmlspy.com)-->
<leveltwo xmlns:gms="GMS_schemas" xmlns:bmt="BMT_schemas" xmlns:cda="KAI_CDA"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="BMT_schemas BMT_Discharge.xsd">
<cda:clinical_document_header>
</cda:clinical_document_header>

<cda:section>
<cda:caption>IBM Genomic Messaging System Data</cda:caption>

<cda:paragraph>
<cda:content>
<cda:local_markup ignore="markup">
<gms:annotation>GMS-augmented document created Sat Dec 1
17:25:59 2001 gms:environment tags allow use of valid xml as annotation mixed
with DNA in ..GATTACCA.. format, and executable GMSL (Genomic Messaging Stream
Language) as content. The GMSL will activate immediately when program gms is
run with the IBM-Yorktown legacy conversion cartridge option selected for IBM-
Haifa CDA hospital files.</gms:annotation>

<gms:genomic_data>
<gms:dna sequence="1" base="1" locus="1">
<gms:gene sequence="HLA00664,DRB1*0101,801 bases, 80C9FCB6
checksum" />

<gms:t_cell_epitopes>
<gms:protein>pkyykqn
<gms:pkc_phosphorylation>tlk</gms:pkc_phosphorylation>

```


Genomic Messaging System

research articles

```

QACGHHPSLLFLAVPEPPDFNAAWIET
<gms:pkc_phosphorylation>THR</gms:pkc_phosphorylation>
ADQEVVVLQGLGL
<gms:protein_feature type="whole_sequence"
sequence="1" context="protein-sized entity" readingframe="2 complement
start="1" stop="133" />
<gms:protein_feature type="gene
sequence=HLA00664,DRB1*0101,801 bases, 80C9FCB6 checksum/" sequence="1"
readingframe="2 complement" start="1" />
<gms:protein_feature type="t_cell_epitopes"
sequence="1" readingframe="2 complement" start="0" stop="0" />
<gms:protein_feature type="protein" sequence="1"
readingframe="2 complement" start="0" stop="0" />
<gms:protein_feature type="protein" sequence="1"
readingframe="2 complement" start="0" stop="0" />
<gms:protein_feature type="st_phosphorylation"
sequence="1" readingframe="2 complement" start="1" stop="4" />
<gms:protein_feature
type="experimental_start_of_mature_peptide/" sequence="1" readingframe="2
complement" start="30" />
<gms:protein_feature type="glycosylation"
sequence="1" readingframe="2 complement" start="69" stop="71" />
<gms:protein_feature type="snp" sequence="1"
readingframe="2 complement" start="72" stop="72" />
<gms:protein_feature type="st_phosphorylation"
sequence="1" readingframe="2 complement" start="87" stop="90" />
<gms:protein_feature type="pkc_phosphorylation"
sequence="1" readingframe="2 complement" start="118" stop="120" />
</gms:translation>
<gms:protein_sequence sequence="1" readingframe="3"
length="40">GVSEAPWRLHDSADSDTDGAEPLTGFQGWGHPHTFLVAA.</gms:protein_sequence>
<gms:translation sequence="1" readingframe="3">
<gms:gene sequence="HLA00664,DRB1*0101,801 bases,
80C9FCB6 checksum" />
<gms:t_cell_epitopes>
<gms:protein>pkyykqntlkla {treat as
peptide}</gms:protein>
<gms:protein>gplkaeiaqrle {treat as
peptide}</gms:protein>
</gms:t_cell_epitopes>
GVSEAPWRLHDSAD
<gms:st_phosphorylation>SDTD</gms:st_phosphorylation>
GAEPLTGFQW
<gms:experimental_start_of_mature_peptide />
GHPTTLVAA
<gms:stop_codon>.</gms:stop_codon>
<gms:protein_feature type="whole_sequence"
sequence="1" context="protein-sized entity" readingframe="3" start="1"
stop="39" />
<gms:protein_feature type="gene
sequence=HLA00664,DRB1*0101,801 bases, 80C9FCB6 checksum/" sequence="1"
readingframe="3" start="1" />
<gms:protein_feature type="t_cell_epitopes"
sequence="1" readingframe="3" start="0" stop="0" />
<gms:protein_feature type="protein" sequence="1"
readingframe="3" start="0" stop="0" />
<gms:protein_feature type="protein" sequence="1"
readingframe="3" start="0" stop="0" />
<gms:protein_feature type="st_phosphorylation"
sequence="1" readingframe="3" start="16" stop="19" />
<gms:protein_feature
type="experimental_start_of_mature_peptide/" sequence="1" readingframe="3"
start="30" />
<gms:protein_feature type="stop_codon" sequence="1"
readingframe="3" start="40" stop="39" />
</gms:translation>
<gms:protein_sequence sequence="1" readingframe="3
complement" length="135">SAQBSVWLSKRSVSLIPEVDEQPGPKBQAQHEAPDSTQHLALCRFRPCE
FHCEARHTWVHLGAVNLSSTPRNCFQHQGLEGGPISILDQACGHHPSLL
LAVPEPPDFNAAWIETTHRADQEVVVLQGLGL.</gms:protein_sequence>
<gms:translation sequence="1" readingframe="3
complement">
<gms:gene sequence="HLA00664,DRB1*0101,801 bases,
80C9FCB6 checksum" />
<gms:t_cell_epitopes>
<gms:protein>pkyykqntlkla {treat as
peptide}</gms:protein>
<gms:protein>gplkaeiaqrle {treat as
peptide}</gms:protein>
</gms:t_cell_epitopes>
<gms:st_phosphorylation>SAQE</gms:st_phosphorylation>
SWLKSRSVSLIPEVDEQPGPKBQ
<gms:experimental_start_of_mature_peptide />
AQHEAPDSTQHLALCRFRPCEFHCEARHTWVHLGAV
<gms:glycosylation>NLS</gms:glycosylation>
<gms:snp>P</gms:snp>
RNCFQHQGLEGGI
<gms:st_phosphorylation>SILD</gms:st_phosphorylation>
QACGHHPSLLFLAVPEPPDFNAAWIET
<gms:pkc_phosphorylation>THR</gms:pkc_phosphorylation>
ADQEVVVLQGLGL
<gms:protein_feature type="whole_sequence"
sequence="1" context="protein-sized entity" readingframe="3 complement"
start="1" stop="133" />
<gms:protein_feature type="gene
sequence=HLA00664,DRB1*0101,801 bases, 80C9FCB6 checksum/" sequence="1"
readingframe="3 complement" start="1" />
<gms:protein_feature type="t_cell_epitopes"
sequence="1" readingframe="3 complement" start="0" stop="0" />
<gms:protein_feature type="protein" sequence="1"
readingframe="3 complement" start="0" stop="0" />
<gms:protein_feature type="protein" sequence="1"
readingframe="3 complement" start="0" stop="0" />
<gms:protein_feature type="st_phosphorylation"
sequence="1" readingframe="3 complement" start="1" stop="4" />
<gms:protein_feature type="protein" sequence="1"
readingframe="3 complement" start="0" stop="0" />
<gms:protein_feature type="st_phosphorylation"
sequence="1" readingframe="3 complement" start="1" stop="4" />
<gms:protein_feature
type="experimental_start_of_mature_peptide/" sequence="1" readingframe="3
complement" start="30" />
<gms:protein_feature type="glycosylation"
sequence="1" readingframe="3 complement" start="69" stop="71" />
<gms:protein_feature type="snp" sequence="1"
readingframe="3 complement" start="72" stop="72" />
<gms:protein_feature type="st_phosphorylation"
sequence="1" readingframe="3 complement" start="87" stop="90" />
<gms:protein_feature type="pkc_phosphorylation"
sequence="1" readingframe="3 complement" start="118" stop="120" />
</gms:translation>
</gms:protein_annotation>
</gms:automated_annotation>
</gms:dna>
<gms:dna sequence="2" base="802" locus="1">
<gms:annotation>possible somatic mutation cell line #4
end-11th</gms:annotation>
AGGAATCAGAAAGGACACTCTGGACTTCAGCCCAACAGGATACCTGAGCTGA
<gms:automated_annotation>
<gms:length sequence="2">51</gms:length>
<gms:bases_so_far sequence="2">51</gms:bases_so_far>
<gms:invalid_dna_symbols>
</gms:invalid_dna_symbols>
<gms:agct_count>a18 g13 c12 t8</gms:agct_count>
<gms:agct_ratio>a35% g25% c24% t16%</gms:agct_ratio>
AGGAATCAGAAAGGACACTCTGGACTTCAGCCCAACAGGATACCTGAGCTGA
<gms:dna_feature type="whole_sequence" sequence="2"
start="1" stop="51" />
<gms:protein_sequence sequence="2" readingframe="1"
length="17">RNQKHSGLQPTGYLS.</gms:protein_sequence>
<gms:protein_annotation>
<gms:translation sequence="2" readingframe="1">RNQKHS
<gms:myristyl>GLQPTG</gms:myristyl>
YLS.
<gms:protein_feature type="whole_sequence" sequence="2"
context="binding peptide?" readingframe="1" start="1" stop="16" />
<gms:protein_feature type="myristyl" sequence="2"
readingframe="1" start="8" stop="13" />
</gms:translation>
<gms:protein_sequence sequence="2" readingframe="1
complement" length="17">SAQVSCWLKSRVSFLIP</gms:protein_sequence>
<gms:translation sequence="2" readingframe="1
complement">SAQVSCWLKSRVSFLIP
<gms:protein_feature type="whole_sequence" sequence="2"
context="binding peptide?" readingframe="1 complement" start="1" stop="17"
/>
</gms:translation>
<gms:protein_sequence sequence="2" readingframe="2"
length="15">GIRKDTLDFSQDD.</gms:protein_sequence>
<gms:translation sequence="2"
readingframe="2">GIRKDTLDF
<gms:st_phosphorylation>SQDD</gms:st_phosphorylation>
T.
<gms:protein_feature type="whole_sequence" sequence="2"
context="binding peptide?" readingframe="2" start="1" stop="14" />
<gms:protein_feature type="st_phosphorylation"
sequence="2" readingframe="2" start="10" stop="13" />
</gms:translation>
<gms:protein_sequence sequence="2" readingframe="2
complement" length="18">SAQVSCWLKSRVSFLIcc</gms:protein_sequence>
<gms:translation sequence="2" readingframe="2
complement">SAQVSCWLKSRVSFLIcc
<gms:protein_feature type="whole_sequence" sequence="2"
context="binding peptide?" readingframe="2 complement" start="1" stop="16"
/>
</gms:translation>

```

research articles

Robson and Mushlin

```

<gms:protein_sequence sequence="2" readingframe="3"
length="17">ESERTLWTSANRIPELa</gms:protein_sequence>

<gms:translation sequence="2" readingframe="3">E
<gms:pkc_phosphorylation>SER</gms:pkc_phosphorylation>

TLWTSANRIPELa
<gms:protein_feature type="whole_sequence" sequence="2"
context="binding peptide?" readingframe="3" start="1" stop="16" />

<gms:protein_feature type="pkc_phosphorylation"
sequence="2" readingframe="3" start="2" stop="4" />
</gms:translation>

<gms:protein_sequence sequence="2" readingframe="3"
complement" length="17">SAQVSCWLKSRVSLIc</gms:protein_sequence>

<gms:translation sequence="2" readingframe="3

```

```

complement">SAQVSCWLKSRVSLIc
<gms:protein_feature type="whole_sequence" sequence="2"
context="binding peptide?" readingframe="3" complement" start="1" stop="16"
/>
</gms:translation>
</gms:protein_annotation>
</gms:automated_annotation>
</gms:dna>

<gms:dna sequence="3" base="853" locus="1">{end}</gms:dna>

<gms:dna_checks>accumulative basepair count = 852,
sequences = 2</gms:dna_checks>
</gms:genomic_data>
</cda:local_markup>
</cda:content>
</cda:paragraph>
</cda:section>
</leveltwo>

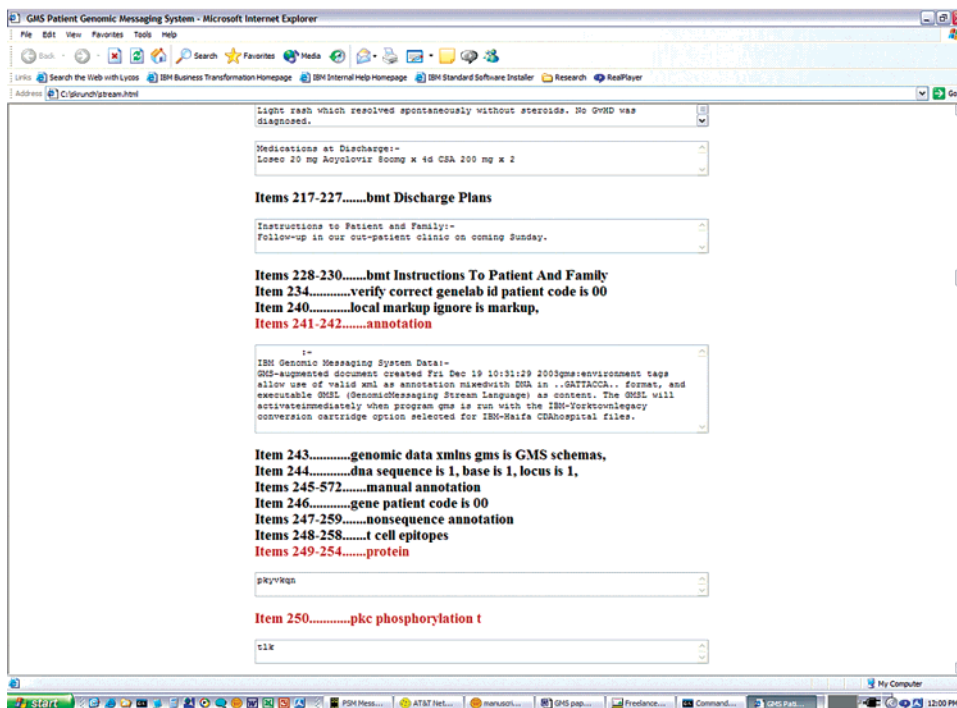
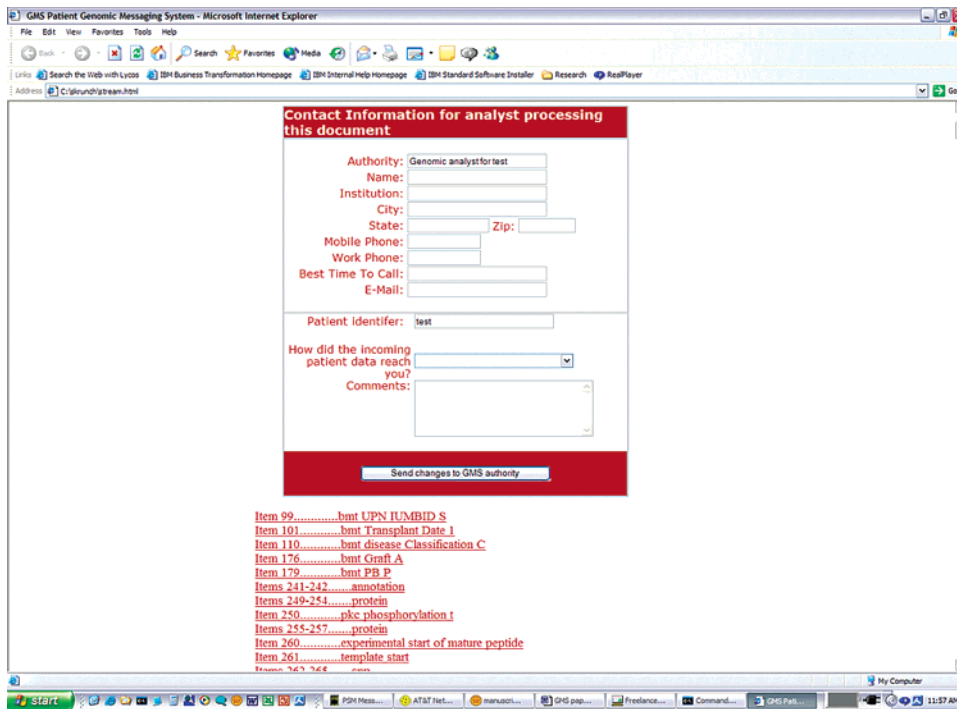
```

1257

Appendix 6

Sample Screen Shots of Display of .html File after Automatic Annotation. CDA, or Clinical Document Architecture is a specific embodiment of XML proposed by Health Level Seven Inc. for medical applications.

1258



Genomic Messaging System

research articles

1259 Note added in Proof

1260 In addition to the commands given in Appendix 1, the special
 1261 command {cc spreadsheet} can appear in any data statement.
 1262 Characters to the left of this command are copied to a line of
 1263 spreadsheet (actually, as a row in comma-separated value
 1264 format). This does not impede the functions of any other
 1265 commands associated with the data statement. The file to
 1266 which the data is written has the suffix .spr and the same root
 1267 name as the user selects for the other files, e.g., a line of
 1268 spreadsheet is written to gms.spr if gms is the choice root name
 1269 for other files. The first time the command is encountered, this
 1270 file is created, and each call adds a new last row. Amongst other
 1271 applications, the spreadsheet file can represent the link to
 1272 FANO clinical data mining software also reported in this journal
 1273 (clinical and pharmacogenomic data mining. 1. The generalized
 1274 theory of expected information and application to the develop-
 1275 ment of tools B. Robson *J. Proteome Res.* **2003**, *2*, 283–301;
 1276 Clinical and pharmacogenomic data mining 2. A Simple
 1277 Method for the Combination of Information from Associations
 1278 and Multivariates to Facilitate Analysis, Decision and Design
 1279 in Clinical Research and Practice. B. Robson and R. Mushlin *J.*
 1280 *Proteome Res.* **2004**, in press.)

1281 References

- 1282 (1) Li, J.; Robson, B. "Bioinformatics and Computational Chemistry
 1283 in molecular Design. Recent Advances and their Application" In
 1284 *Peptide and Protein Drug Analysis*; Marcel Dekker: New York;
 1285 2000, pp 285–307.
 1286 (2) Robson B.; Garnier J. "The future of highly personalized health
 1287 care" *Stud. Health Technol. Info.* **2002**, *80*, 163–174.
 1288 (3) Robson, B.; Greaney, P. J. "Natural Sequence Code Representa-
 1289 tions for Compression and Rapid Searching of Human-Genome
 1290 Style Data Base", *CABIOS* **1992**, *8*, 283–289.
 1291 (4) Robson, B. "Studies in the Assessment of Folding Quality for
 1292 Protein Modeling and Simulation when the Experimental Struc-
 1293 ture is Unknown". *J. Proteome Res.* **2002**, *1*(2), 115–133.
 1294 (5) Ball, J.; Fishleigh, R. V.; Greaney, P. J.; Marsden, A.; Platt, E.; Pool,
 1295 J. L.; Robson, B. "A Polymorphic Programming Environment for
 1296 the Chemical Pharmaceutical and Biotechnology Industries in

- 'Chemical Information Systems – Beyond the Structure Dia- 1297
 grams.' Bawden, D., Mitchell, E. M., Eds., Ellis Horwood Press: 1298
 Chichester, 1990; 107–123. 1299
 (6) Platt, E.; Robson, B. "Case Studies in Automatic Modelling of 1300
 Thrombin, Alpha₂-Macroglobulin and other Proteins, and Implica- 1301
 tions for Drug Design", *Proc. R. Soc. Edinburgh* **1992**, *99B*, (1/2), 1302
 123–136. 1303
 (7) Collura, V. P.; Greaney, P. J.; Robson, B. "A Method for Rapidly 1304
 Assessing and Refining Simple Solvent Treatments in Molecular 1305
 Modelling. Example Studies on the Antigen-Combining Loop H2 1306
 from FAB Fragment McPC603", *Protein Eng.* **1990**, *7*, 221–233. 1307
 (8) Clark, D. E.; Frenkel, D.; Levy, S. A.; Murray, C. W.; Robson, B.; 1308
 Waszkowycz, B.; Westhead, D. R. "PRO LIGAND: An Approach 1309
 to De Novo Molecular Design. 1. Application to the Design of 1310
 Organic Molecules", *J. Comput. Aided Mol. Des.* **1995**, *9*, 13–32. 1311
 (9) Waszkowycz, B.; Clark, D. E.; Frenkel, D.; Li, J.; Murray, C. W.; 1312
 Robson B.; Westhead, D. R. "PRO LIGAND: An Approach to De 1313
 Novo Design. 2. Design of Novel Molecules from Molecular Field 1314
 Analysis (MFA) Models and Pharmacophores", *J. Med. Chem.* 1315
1994, *37*, 3994–4002. 1316
 (10) Westhead, D. R.; Clark, D. E.; Frenkel, D.; Li, J.; Murray, C. W.; 1317
 Robson, B.; Waszkowycz, B. "PRO LIGAND: An Approach to De 1318
 Novo Molecular Design. 3. A Genetic Algorithm for Structure 1319
 Refinement". *J. Comput. Aided Mol. Des.* **1995**, *9*, 139–148. 1320
 (11) Frenkel, D.; Clark, D. E.; Li, J.; Murray, C. W.; Robson, B.; 1321
 Waszkowycz, B.; Westhead, D. R. "PRO LIGAND: An Approach 1322
 to De Novo Molecular Design. 4. Application to the Design of 1323
 Peptides", *J. Comput. Aided Mol. Des.* **1995**, *9*, 213–225. 1324
 (12) Robson, B. Circulated Report from NIH Symposium Protein 1325
 Folding and Design comment reported in Chemical Engineering 1326
 News, May 27, 1996, pp 32–33. 1327
 (13) Robson, B. "Doppelganger Proteins as Drug Leads", *Nat. Bio-* 1328
technol. **1996**, *14*, 892–893. 1329
 (14) Figliozzi, G. M.; Siani, M. A.; Canne, L. E.; Robson, B.; Simon, R. 1330
 J. "Chemical synthesis and activity of D-superoxide dismutase", 1331
Protein Science, *5*, Suppl.1, 72, [published poster 87-S]; 1996, 1332
 included in "Total Chemical Synthesis of Proteins including 1333
 Chemokines and their Analogues", Siani, M. A.; Canne, L. E.; 1334
 Figliozzi, G. M.; Robson, B.; Thompson D. A.; Simon, R. J. 1335
 circulated in "Chemokines", International Business Communica- 1336
 tions, 1997. 1337
 (15) Robson, B. "Fast Track Routes from Genome to Peptidomimetic" 1338
 In *Peptidomimetics & Small Molecule Design*; Hori, W., Nagle, E. 1339
 M.; Savage, L. M. 1998. 1340
 (16) Robson, B. "Beyond Proteins" Robson, B. *Trends Biotechnol.* **1999**, 1341
17, 311–315. 1342

PR0341336

1343