

INTERNATIONAL STANDARD

ISO/IEC
11756

First edition
1992-12-15

Information technology – Programming languages – MUMPS

Technologies de l'information – Langages de programmation – MUMPS



Reference number
ISO/IEC 11756:1992 (E)
Documents
Document Sharing Hub

Table of Contents

Part 1: MUMPS Language Specification

1 Static Syntax Metalanguage 2

2 Static Syntax and Semantics 3

 2.1 Basic Alphabet 3

 2.2 Expression Atom expratom 3

 2.2.1 Name name 4

 2.2.2 Variables 4

 2.2.2.1 Local Variable Name lvn 4

 2.2.2.2 Global Variable Name gvn 5

 2.2.2.3 Variable Handling 6

 2.2.2.4 Variable Contexts 9

 2.2.3 Numeric Literal numlit 9

 2.2.3.1 Numeric Data Values 10

 2.2.3.2 Meaning of numlit 10

 2.2.4 Numeric Interpretation of Data 11

 2.2.4.1 Integer Interpretation 12

 2.2.4.2 Truth-Value Interpretation 12

 2.2.5 String Literal strlit 12

 2.2.6 Intrinsic Special Variable Name svn 13

 2.2.7 Intrinsic Functions function 15

 2.2.7.1 \$ASCII 15

 2.2.7.2 \$CHAR 16

 2.2.7.3 \$DATA 16

 2.2.7.4 \$EXTRACT 17

 2.2.7.5 \$FIND 17

 2.2.7.6 \$FNUMBER 18

 2.2.7.7 \$GET 19

 2.2.7.8 \$JUSTIFY 19

 2.2.7.9 \$LENGTH 20

 2.2.7.10 \$NEXT 20

 2.2.7.11 \$ORDER 21

 2.2.7.12 \$PIECE 21

 2.2.7.13 \$QUERY 22

 2.2.7.14 \$RANDOM 24

2.2.7.15 \$SELECT	24
2.2.7.16 \$TEXT	24
2.2.7.18 \$VIEW	25
2.2.7.19 \$Z	25
2.2.8 Unary Operator <u>unaryop</u>	25
2.2.9 Extrinsic Special Variable	25
2.2.10 Extrinsic Function	26
2.3 Expressions <u>expr</u>	26
2.3.1 Arithmetic Binary Operators	27
2.3.2 Relational Operators	27
2.3.2.1 Numeric Relations	28
2.3.2.2 String Relations	28
2.3.3 Pattern match	28
2.3.4 Logical Operators	29
2.3.5 Concatenation Operator	30
2.4 Routines	31
2.4.1 Routine Structure	31
2.4.2 Routine Execution	31
2.5 General <u>command</u> Rules	32
2.5.1 Post Conditionals	33
2.5.2 Spaces in Commands	34
2.5.3 Comments	34
2.5.4 <u>format</u> in READ and WRITE	34
2.5.5 Side Effects on \$X and \$Y	34
2.5.6 Timeout	35
2.5.7 Line References	35
2.5.8 Command Argument Indirection	36
2.5.9 Parameter Passing	37
2.6 Command Definitions	37
2.6.1 BREAK	39
2.6.2 CLOSE	39
2.6.3 DO	39
2.6.4 ELSE	40
2.6.5 FOR	41
2.6.6 GOTO	41
2.6.7 HALT	43
2.6.8 HANG	43
2.6.9 IF	43
2.6.10 JOB	44
2.6.11 KILL	44
2.6.12 LOCK	45
2.6.13 NEW	46
2.6.14 OPEN	48
2.6.15 QUIT	49
2.6.16 READ	49
2.6.17 SET	51
2.6.18 USE	52
2.6.19 VIEW	53
2.6.20 WRITE	54
2.6.21 XECUTE	54
2.6.22 Z	55

Part 2: MUMPS Portability requirements

Introduction 61

1 Expression Elements 62

 1.1 Names 62

 1.2 Local Variables 62

 1.3 Global Variables 62

 1.4 Data Types 63

 1.5 Number Range 63

 1.6 Integers 64

 1.7 Character Strings 64

 1.8 Special Variables 64

2 Expressions 64

 2.1 Nesting of Expressions 64

 2.2 Results 64

3 Routines and Command Lines 64

 3.1 Command Lines 64

 3.2 Number of Command Lines 65

 3.3 Number of Commands 65

 3.4 Labels 65

 3.5 Number of Labels 65

 3.6 Number of Routines 65

4 Indirection 65

5 Storage Space Restrictions 65

6 Nesting 66

7 Other Portability Requirements 66

Appendix A: ASCII Character Set (ANSI X3.4-1986) 67

Appendix B: Metalanguage Elements 71

Index 81

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 11756 was prepared by American National Standards Institute (ANSI) (as ANSI/MDC X11.1-1990) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Appendices A and B of this International Standard are for information only.

Terminology and conventions

The text of American National Standard Institute ANSI/MDC X11.1-1990 has been approved for publication, without deviation, as an International Standard. Some terminology and certain conventions are not in accordance with the ISO/IEC Directives Part 3: "Drafting and presentation of International Standards"; attention is especially drawn to the following:

Wherever the word "standard" appears, referring to this International Standard, it should be read as "International Standard".

Cross reference

American National Standard	Corresponding International Standard
ANSI X3.4-1986	ISO/IEC 646:1991, <i>Information technology – ISO 7-bit coded character set for information interchange</i> .

1. INTRODUCTION

2. REFERENCES

3. DEFINITIONS

4. SCOPE

5. GENERAL PRINCIPLES

6. TEST METHODS

7. EVALUATION

8. REPORTING

9. APPENDICES

10. BIBLIOGRAPHY

11. ANNEXES

12. INDEX

13. GLOSSARY

14. LIST OF FIGURES

15. LIST OF TABLES

16. SUMMARY

17. CONCLUSIONS

18. RECOMMENDATIONS

19. REFERENCES

20. DEFINITIONS

21. SCOPE

22. GENERAL PRINCIPLES

23. TEST METHODS

24. EVALUATION

25. REPORTING

26. APPENDICES

27. BIBLIOGRAPHY

28. ANNEXES

29. INDEX

30. GLOSSARY

31. LIST OF FIGURES

32. LIST OF TABLES

33. SUMMARY

34. CONCLUSIONS

35. RECOMMENDATIONS

36. REFERENCES

37. DEFINITIONS

38. SCOPE

39. GENERAL PRINCIPLES

40. TEST METHODS

41. EVALUATION

42. REPORTING

43. APPENDICES

44. BIBLIOGRAPHY

45. ANNEXES

46. INDEX

47. GLOSSARY

48. LIST OF FIGURES

49. LIST OF TABLES

50. SUMMARY

51. CONCLUSIONS

52. RECOMMENDATIONS

53. REFERENCES

54. DEFINITIONS

55. SCOPE

56. GENERAL PRINCIPLES

57. TEST METHODS

58. EVALUATION

59. REPORTING

60. APPENDICES

61. BIBLIOGRAPHY

62. ANNEXES

63. INDEX

64. GLOSSARY

65. LIST OF FIGURES

66. LIST OF TABLES

67. SUMMARY

68. CONCLUSIONS

69. RECOMMENDATIONS

70. REFERENCES

71. DEFINITIONS

72. SCOPE

73. GENERAL PRINCIPLES

74. TEST METHODS

75. EVALUATION

76. REPORTING

77. APPENDICES

78. BIBLIOGRAPHY

79. ANNEXES

80. INDEX

81. GLOSSARY

82. LIST OF FIGURES

83. LIST OF TABLES

84. SUMMARY

85. CONCLUSIONS

86. RECOMMENDATIONS

87. REFERENCES

88. DEFINITIONS

89. SCOPE

90. GENERAL PRINCIPLES

91. TEST METHODS

92. EVALUATION

93. REPORTING

94. APPENDICES

95. BIBLIOGRAPHY

96. ANNEXES

97. INDEX

98. GLOSSARY

99. LIST OF FIGURES

100. LIST OF TABLES

101. SUMMARY

102. CONCLUSIONS

103. RECOMMENDATIONS

104. REFERENCES

105. DEFINITIONS

106. SCOPE

107. GENERAL PRINCIPLES

108. TEST METHODS

109. EVALUATION

110. REPORTING

111. APPENDICES

112. BIBLIOGRAPHY

113. ANNEXES

114. INDEX

115. GLOSSARY

116. LIST OF FIGURES

117. LIST OF TABLES

118. SUMMARY

119. CONCLUSIONS

120. RECOMMENDATIONS

121. REFERENCES

122. DEFINITIONS

123. SCOPE

124. GENERAL PRINCIPLES

125. TEST METHODS

126. EVALUATION

127. REPORTING

128. APPENDICES

129. BIBLIOGRAPHY

130. ANNEXES

131. INDEX

132. GLOSSARY

133. LIST OF FIGURES

134. LIST OF TABLES

135. SUMMARY

136. CONCLUSIONS

137. RECOMMENDATIONS

138. REFERENCES

139. DEFINITIONS

140. SCOPE

141. GENERAL PRINCIPLES

142. TEST METHODS

143. EVALUATION

144. REPORTING

145. APPENDICES

146. BIBLIOGRAPHY

147. ANNEXES

148. INDEX

149. GLOSSARY

150. LIST OF FIGURES

151. LIST OF TABLES

152. SUMMARY

153. CONCLUSIONS

154. RECOMMENDATIONS

155. REFERENCES

156. DEFINITIONS

157. SCOPE

158. GENERAL PRINCIPLES

159. TEST METHODS

160. EVALUATION

161. REPORTING

162. APPENDICES

163. BIBLIOGRAPHY

164. ANNEXES

165. INDEX

166. GLOSSARY

167. LIST OF FIGURES

168. LIST OF TABLES

169. SUMMARY

170. CONCLUSIONS

171. RECOMMENDATIONS

172. REFERENCES

173. DEFINITIONS

174. SCOPE

175. GENERAL PRINCIPLES

176. TEST METHODS

177. EVALUATION

178. REPORTING

179. APPENDICES

180. BIBLIOGRAPHY

181. ANNEXES

182. INDEX

183. GLOSSARY

184. LIST OF FIGURES

185. LIST OF TABLES

186. SUMMARY

187. CONCLUSIONS

188. RECOMMENDATIONS

189. REFERENCES

190. DEFINITIONS

191. SCOPE

192. GENERAL PRINCIPLES

193. TEST METHODS

194. EVALUATION

195. REPORTING

196. APPENDICES

197. BIBLIOGRAPHY

198. ANNEXES

199. INDEX

200. GLOSSARY

201. LIST OF FIGURES

202. LIST OF TABLES

203. SUMMARY

204. CONCLUSIONS

205. RECOMMENDATIONS

206. REFERENCES

207. DEFINITIONS

208. SCOPE

209. GENERAL PRINCIPLES

210. TEST METHODS

211. EVALUATION

212. REPORTING

213. APPENDICES

214. BIBLIOGRAPHY

215. ANNEXES

216. INDEX

217. GLOSSARY

218. LIST OF FIGURES

219. LIST OF TABLES

220. SUMMARY

221. CONCLUSIONS

222. RECOMMENDATIONS

223. REFERENCES

224. DEFINITIONS

225. SCOPE

226. GENERAL PRINCIPLES

227. TEST METHODS

228. EVALUATION

229. REPORTING

230. APPENDICES

231. BIBLIOGRAPHY

232. ANNEXES

233. INDEX

234. GLOSSARY

235. LIST OF FIGURES

236. LIST OF TABLES

237. SUMMARY

238. CONCLUSIONS

239. RECOMMENDATIONS

240. REFERENCES

241. DEFINITIONS

242. SCOPE

243. GENERAL PRINCIPLES

244. TEST METHODS

245. EVALUATION

246. REPORTING

247. APPENDICES

248. BIBLIOGRAPHY

249. ANNEXES

250. INDEX

251. GLOSSARY

252. LIST OF FIGURES

253. LIST OF TABLES

254. SUMMARY

255. CONCLUSIONS

256. RECOMMENDATIONS

257. REFERENCES

258. DEFINITIONS

259. SCOPE

260. GENERAL PRINCIPLES

261. TEST METHODS

262. EVALUATION

263. REPORTING

264. APPENDICES

265. BIBLIOGRAPHY

266. ANNEXES

267. INDEX

268. GLOSSARY

269. LIST OF FIGURES

270. LIST OF TABLES

271. SUMMARY

272. CONCLUSIONS

273. RECOMMENDATIONS

274. REFERENCES

275. DEFINITIONS

276. SCOPE

277. GENERAL PRINCIPLES

278. TEST METHODS

279. EVALUATION

280. REPORTING

281. APPENDICES

282. BIBLIOGRAPHY

283. ANNEXES

284. INDEX

285. GLOSSARY

286. LIST OF FIGURES

287. LIST OF TABLES

288. SUMMARY

289. CONCLUSIONS

290. RECOMMENDATIONS

291. REFERENCES

292. DEFINITIONS

293. SCOPE

294. GENERAL PRINCIPLES

295. TEST METHODS

296. EVALUATION

297. REPORTING

298. APPENDICES

299. BIBLIOGRAPHY

300. ANNEXES

301. INDEX

302. GLOSSARY

303. LIST OF FIGURES

304. LIST OF TABLES

305. SUMMARY

306. CONCLUSIONS

307. RECOMMENDATIONS

308. REFERENCES

309. DEFINITIONS

310. SCOPE

311. GENERAL PRINCIPLES

312. TEST METHODS

313. EVALUATION

314. REPORTING

315. APPENDICES

316. BIBLIOGRAPHY

317. ANNEXES

318. INDEX

319. GLOSSARY

320. LIST OF FIGURES

321. LIST OF TABLES

322. SUMMARY

323. CONCLUSIONS

324. RECOMMENDATIONS

325. REFERENCES

326. DEFINITIONS

327. SCOPE

328. GENERAL PRINCIPLES

329. TEST METHODS

330. EVALUATION

331. REPORTING

332. APPENDICES

333. BIBLIOGRAPHY

334. ANNEXES

335. INDEX

336. GLOSSARY

337. LIST OF FIGURES

338. LIST OF TABLES

339. SUMMARY

340. CONCLUSIONS

341. RECOMMENDATIONS

342. REFERENCES

343. DEFINITIONS

344. SCOPE

345. GENERAL PRINCIPLES

346. TEST METHODS

347. EVALUATION

348. REPORTING

349. APPENDICES

350. BIBLIOGRAPHY

351. ANNEXES

352. INDEX

353. GLOSSARY

354. LIST OF FIGURES

355. LIST OF TABLES

356. SUMMARY

357. CONCLUSIONS

358. RECOMMENDATIONS

359. REFERENCES

360. DEFINITIONS

361. SCOPE

362. GENERAL PRINCIPLES

363. TEST METHODS

364. EVALUATION

365. REPORTING

366. APPENDICES

367. BIBLIOGRAPHY

368. ANNEXES

369. INDEX

370. GLOSSARY

371. LIST OF FIGURES

372. LIST OF TABLES

373. SUMMARY

374. CONCLUSIONS

375. RECOMMENDATIONS

376. REFERENCES

377. DEFINITIONS

378. SCOPE

379. GENERAL PRINCIPLES

380. TEST METHODS

381. EVALUATION

382. REPORTING

383. APPENDICES

384. BIBLIOGRAPHY

385. ANNEXES

386. INDEX

387. GLOSSARY

388. LIST OF FIGURES

389. LIST OF TABLES

390. SUMMARY

391. CONCLUSIONS

392. RECOMMENDATIONS

393. REFERENCES

394. DEFINITIONS

395. SCOPE

396. GENERAL PRINCIPLES

397. TEST METHODS

398. EVALUATION

399. REPORTING

400. APPENDICES

401. BIBLIOGRAPHY

402. ANNEXES

403. INDEX

404. GLOSSARY

405. LIST OF FIGURES

406. LIST OF TABLES

407. SUMMARY

408. CONCLUSIONS

409. RECOMMENDATIONS

410. REFERENCES

411. DEFINITIONS

412. SCOPE

413. GENERAL PRINCIPLES

414. TEST METHODS

415. EVALUATION

416. REPORTING

417. APPENDICES

418. BIBLIOGRAPHY

419. ANNEXES

420. INDEX

421. GLOSSARY

422. LIST OF FIGURES

423. LIST OF TABLES

424. SUMMARY

425. CONCLUSIONS

426. RECOMMENDATIONS

427. REFERENCES

428. DEFINITIONS

429. SCOPE

430. GENERAL PRINCIPLES

431. TEST METHODS

432. EVALUATION

433. REPORTING

434. APPENDICES

435. BIBLIOGRAPHY

436. ANNEXES

437. INDEX

438. GLOSSARY

439. LIST OF FIGURES

440. LIST OF TABLES

441. SUMMARY

442. CONCLUSIONS

443. RECOMMENDATIONS

444. REFERENCES

445. DEFINITIONS

446. SCOPE

447. GENERAL PRINCIPLES

448. TEST METHODS

449. EVALUATION

450. REPORTING

451. APPENDICES

452. BIBLIOGRAPHY

453. ANNEXES

454. INDEX

455. GLOSSARY

456. LIST OF FIGURES

457. LIST OF TABLES

458. SUMMARY

459. CONCLUSIONS

460. RECOMMENDATIONS

461. REFERENCES

462. DEFINITIONS

463. SCOPE

464. GENERAL PRINCIPLES

465. TEST METHODS

466. EVALUATION

467. REPORTING

468. APPENDICES

469. BIBLIOGRAPHY

470. ANNEXES

471. INDEX

472. GLOSSARY

473. LIST OF FIGURES

474. LIST OF TABLES

475. SUMMARY

476. CONCLUSIONS

477. RECOMMENDATIONS

478. REFERENCES

479. DEFINITIONS

480. SCOPE

481. GENERAL PRINCIPLES

482. TEST METHODS

483. EVALUATION

484. REPORTING

485. APPENDICES

486. BIBLIOGRAPHY

487. ANNEXES

488. INDEX

489. GLOSSARY

490. LIST OF FIGURES

491. LIST OF TABLES

492. SUMMARY

493. CONCLUSIONS

494. RECOMMENDATIONS

495. REFERENCES

496. DEFINITIONS

497. SCOPE

498. GENERAL PRINCIPLES

499. TEST METHODS

500. EVALUATION

501. REPORTING

502. APPENDICES

503. BIBLIOGRAPHY

504. ANNEXES

505. INDEX

506. GLOSSARY

507. LIST OF FIGURES

508. LIST OF TABLES

509. SUMMARY

510. CONCLUSIONS

511. RECOMMENDATIONS

512. REFERENCES

513. DEFINITIONS

514. SCOPE

515. GENERAL PRINCIPLES

516. TEST METHODS

517. EVALUATION

518. REPORTING

519. APPENDICES

520. BIBLIOGRAPHY

521. ANNEXES

522. INDEX

523. GLOSSARY

524. LIST OF FIGURES

525. LIST OF TABLES

526. SUMMARY

527. CONCLUSIONS

528. RECOMMENDATIONS

529. REFERENCES

530. DEFINITIONS

531. SCOPE

532. GENERAL PRINCIPLES

533. TEST METHODS

534. EVALUATION

535. REPORTING

536. APPENDICES

537. BIBLIOGRAPHY

538. ANNEXES

539. INDEX

540. GLOSSARY

541. LIST OF FIGURES

542. LIST OF TABLES

543. SUMMARY

544. CONCLUSIONS

545. RECOMMENDATIONS

546. REFERENCES

547. DEFINITIONS

548. SCOPE

549. GENERAL PRINCIPLES

550. TEST METHODS

551. EVALUATION

552. REPORTING

553. APPENDICES

554. BIBLIOGRAPHY

555. ANNEXES

556. INDEX

557. GLOSSARY

558. LIST OF FIGURES

559. LIST OF TABLES

560. SUMMARY

561. CONCLUSIONS

562. RECOMMENDATIONS

563. REFERENCES

564. DEFINITIONS

565. SCOPE

566. GENERAL PRINCIPLES

567. TEST METHODS

568. EVALUATION

569. REPORTING

570. APPENDICES

571. BIBLIOGRAPHY

572. ANNEXES

573. INDEX

574. GLOSSARY

575. LIST OF FIGURES

576. LIST OF TABLES

577. SUMMARY

578. CONCLUSIONS

579. RECOMMENDATIONS

580. REFERENCES

581. DEFINITIONS

582. SCOPE

583. GENERAL PRINCIPLES

584. TEST METHODS

585. EVALUATION

586. REPORTING

587. APPENDICES

588. BIBLIOGRAPHY

589. ANNEXES

590. INDEX

591. GLOSSARY

592. LIST OF FIGURES

593. LIST OF TABLES

594. SUMMARY

595. CONCLUSIONS

596. RECOMMENDATIONS

597. REFERENCES

598. DEFINITIONS

599. SCOPE

600. GENERAL PRINCIPLES

601. TEST METHODS

602. EVALUATION

603. REPORTING

604. APPENDICES

605. BIBLIOGRAPHY

606. ANNEXES

607. INDEX

608. GLOSSARY

609. LIST OF FIGURES

610. LIST OF TABLES

611. SUMMARY

612. CONCLUSIONS

613. RECOMMENDATIONS

614. REFERENCES

615. DEFINITIONS

616. SCOPE

617. GENERAL PRINCIPLES

618. TEST METHODS

619. EVALUATION

620. REPORTING

621. APPENDICES

622. BIBLIOGRAPHY

623. ANNEXES

624. INDEX

625. GLOSSARY

626. LIST OF FIGURES

627. LIST OF TABLES

628. SUMMARY

629. CONCLUSIONS

630. RECOMMENDATIONS

631. REFERENCES

632. DEFINITIONS

633. SCOPE

634. GENERAL PRINCIPLES

635. TEST METHODS

636. EVALUATION

637. REPORTING

638. APPENDICES

639. BIBLIOGRAPHY

640. ANNEXES

641. INDEX

642. GLOSSARY

643. LIST OF FIGURES

644. LIST OF TABLES

645. SUMMARY

646. CONCLUSIONS

647. RECOMMENDATIONS

648. REFERENCES

649. DEFINITIONS

650. SCOPE

651. GENERAL PRINCIPLES

652. TEST METHODS

653. EVALUATION

654. REPORTING

655. APPENDICES

656. BIBLIOGRAPHY

657. ANNEXES

658. INDEX

659. GLOSSARY

660. LIST OF FIGURES

661. LIST OF TABLES

662. SUMMARY

663. CONCLUSIONS

664. RECOMMENDATIONS

665. REFERENCES

666. DEFINITIONS

667. SCOPE

668. GENERAL PRINCIPLES

669. TEST METHODS

670. EVALUATION

671. REPORTING

672. APPENDICES

673. BIBLIOGRAPHY

674. ANNEXES

675. INDEX

676. GLOSSARY

677. LIST OF FIGURES

678. LIST OF TABLES

679. SUMMARY

680. CONCLUSIONS

681. RECOMMENDATIONS

682. REFERENCES

683. DEFINITIONS

684. SCOPE

685. GENERAL PRINCIPLES

686. TEST METHODS

687. EVALUATION

688. REPORTING

689. APPENDICES

690. BIBLIOGRAPHY

691. ANNEXES

692. INDEX

693. GLOSSARY

694. LIST OF FIGURES

695. LIST OF TABLES

696. SUMMARY

697. CONCLUSIONS

698. RECOMMENDATIONS

699. REFERENCES

700. DEFINITIONS

701. SCOPE

702. GENERAL PRINCIPLES

703. TEST METHODS

704. EVALUATION

705. REPORTING

706. APPENDICES

707. BIBLIOGRAPHY

708. ANNEXES

709. INDEX

710. GLOSSARY

711. LIST OF FIGURES

712. LIST OF TABLES

713. SUMMARY

714. CONCLUSIONS

715. RECOMMENDATIONS

716. REFERENCES

717. DEFINITIONS

718. SCOPE

719. GENERAL PRINCIPLES

720. TEST METHODS

721. EVALUATION

722. REPORTING

723. APPENDICES

724. BIBLIOGRAPHY

725. ANNEXES

726. INDEX

727. GLOSSARY

728. LIST OF FIGURES

729. LIST OF TABLES

730. SUMMARY

731. CONCLUSIONS

732. RECOMMENDATIONS

733. REFERENCES

734. DEFINITIONS

735. SCOPE

736. GENERAL PRINCIPLES

737. TEST METHODS

738. EVALUATION

739. REPORTING

740. APPENDICES

741. BIBLIOGRAPHY

742. ANNEXES

743. INDEX

744. GLOSSARY

745. LIST OF FIGURES

746. LIST OF TABLES

747. SUMMARY

748. CONCLUSIONS

749. RECOMMENDATIONS

750. REFERENCES

751. DEFINITIONS

752. SCOPE

753. GENERAL PRINCIPLES

754. TEST METHODS

755. EVALUATION

756. REPORTING

757. APPENDICES

758. BIBLIOGRAPHY

759. ANNEXES

760. INDEX

761. GLOSSARY

762. LIST OF FIGURES

763. LIST OF TABLES

764. SUMMARY

765. CONCLUSIONS

766. RECOMMENDATIONS

767. REFERENCES

768. DEFINITIONS

769. SCOPE

770. GENERAL PRINCIPLES

771. TEST METHODS

772. EVALUATION

773. REPORTING

774. APPENDICES

775. BIBLIOGRAPHY

776. ANNEXES

777. INDEX

778. GLOSSARY

779. LIST OF FIGURES

780. LIST OF TABLES

781. SUMMARY

782. CONCLUSIONS

783. RECOMMENDATIONS

784. REFERENCES

785. DEFINITIONS

786. SCOPE

787. GENERAL PRINCIPLES

788. TEST METHODS

789. EVALUATION

790. REPORTING

791. APPENDICES

792. BIBLIOGRAPHY

793. ANNEXES

794. INDEX

795. GLOSSARY

796. LIST OF FIGURES

797. LIST OF TABLES

798. SUMMARY

799. CONCLUSIONS

800. RECOMMENDATIONS

801. REFERENCES

802. DEFINITIONS

803. SCOPE

804. GENERAL PRINCIPLES

805. TEST METHODS

806. EVALUATION

807. REPORTING

808. APPENDICES

809. BIBLIOGRAPHY

810. ANNEXES

811. INDEX

812. GLOSSARY

813. LIST OF FIGURES

814. LIST OF TABLES

815. SUMMARY

816. CONCLUSIONS

817. RECOMMENDATIONS

818. REFERENCES

819. DEFINITIONS

820. SCOPE

821. GENERAL PRINCIPLES

822. TEST METHODS

823. EVALUATION

824. REPORTING

825. APPENDICES

826. BIBLIOGRAPHY

827. ANNEXES

828. INDEX

829. GLOSSARY

830. LIST OF FIGURES

831. LIST OF TABLES

832. SUMMARY

833. CONCLUSIONS

834. RECOMMENDATIONS

835. REFERENCES

836. DEFINITIONS

837. SCOPE

838. GENERAL PRINCIPLES

839. TEST METHODS

840. EVALUATION

841. REPORTING

842. APPENDICES

843. BIBLIOGRAPHY

844. ANNEXES

845. INDEX

846. GLOSSARY

847. LIST OF FIGURES

848. LIST OF TABLES

849. SUMMARY

850. CONCLUSIONS

851. RECOMMENDATIONS

852. REFERENCES

853. DEFINITIONS

854. SCOPE

855. GENERAL PRINCIPLES

856. TEST METHODS

857. EVALUATION

858. REPORTING

859. APPENDICES

860. BIBLIOGRAPHY

861. ANNEXES

862. INDEX

863. GLOSSARY

864. LIST OF FIGURES

865. LIST OF TABLES

866. SUMMARY

867. CONCLUSIONS

868. RECOMMENDATIONS

869. REFERENCES

870. DEFINITIONS

871. SCOPE

872. GENERAL PRINCIPLES

873. TEST METHODS

874. EVALUATION

875. REPORTING

876. APPENDICES

877. BIBLIOGRAPHY

878. ANNEXES

879. INDEX

880. GLOSSARY

881. LIST OF FIGURES

882. LIST OF TABLES

883. SUMMARY

884. CONCLUSIONS

885. RECOMMENDATIONS

886. REFERENCES

887. DEFINITIONS

888. SCOPE

889. GENERAL PRINCIPLES

890. TEST METHODS

891. EVALUATION

892. REPORTING

893. APPENDICES

894. BIBLIOGRAPHY

895. ANNEXES

896. INDEX

897. GLOSSARY

898. LIST OF FIGURES

899. LIST OF TABLES

900. SUMMARY

901. CONCLUSIONS

902. RECOMMENDATIONS

903. REFERENCES

904. DEFINITIONS

905. SCOPE

906. GENERAL PRINCIPLES

907. TEST METHODS

908. EVALUATION

909. REPORTING

910. APPENDICES

911. BIBLIOGRAPHY

912. ANNEXES

913. INDEX

914. GLOSSARY

915. LIST OF FIGURES

916. LIST OF TABLES

917. SUMMARY

918. CONCLUSIONS

919. RECOMMENDATIONS

920. REFERENCES

921. DEFINITIONS

922. SCOPE

923. GENERAL PRINCIPLES

924. TEST METHODS

925. EVALUATION

926. REPORTING

927. APPENDICES

928. BIBLIOGRAPHY

929. ANNEXES

930. INDEX

931. GLOSSARY

932. LIST OF FIGURES

933. LIST OF TABLES

934. SUMMARY

935. CONCLUSIONS

936. RECOMMENDATIONS

937. REFERENCES

938. DEFINITIONS

939. SCOPE

940. GENERAL PRINCIPLES

941. TEST METHODS

942. EVALUATION

943. REPORTING

944. APPENDICES

945. BIBLIOGRAPHY

946. ANNEXES

947. INDEX

948. GLOSSARY

949. LIST OF FIGURES

950. LIST OF TABLES

951. SUMMARY

952. CONCLUSIONS

953. RECOMMENDATIONS

954. REFERENCES

955. DEFINITIONS

956. SCOPE

957. GENERAL PRINCIPLES

958. TEST METHODS

959. EVALUATION

960. REPORTING

961. APPENDICES

962. BIBLIOGRAPHY

963. ANNEXES

964. INDEX

965. GLOSSARY

966. LIST OF FIGURES

967. LIST OF TABLES

968. SUMMARY

969. CONCLUSIONS

970. RECOMMENDATIONS

971. REFERENCES

972. DEFINITIONS

973. SCOPE

974. GENERAL PRINCIPLES

975. TEST METHODS

976. EVALUATION

977. REPORTING

978. APPENDICES

979. BIBLIOGRAPHY

980. ANNEXES

981. INDEX

982. GLOSSARY

983. LIST OF FIGURES

984. LIST OF TABLES

985. SUMMARY

986. CONCLUSIONS

987. RECOMMENDATIONS

988. REFERENCES

989. DEFINITIONS

990. SCOPE

991. GENERAL PRINCIPLES

992. TEST METHODS

993. EVALUATION

994. REPORTING

995. APPENDICES

996. BIBLIOGRAPHY

997. ANNEXES

998. INDEX

999. GLOSSARY

1000. LIST OF FIGURES

1001. LIST OF TABLES

1002. SUMMARY

1003. CONCLUSIONS

1004. RECOMMENDATIONS

1005. REFERENCES

1006. DEFINITIONS

1007. SCOPE

1008. GENERAL PRINCIPLES

1009. TEST METHODS

1010. EVALUATION

1011. REPORTING

1012. APPENDICES

1013. BIBLIOGRAPHY

1014. ANNEXES

1015. INDEX

1016. GLOSSARY

1017. LIST OF FIGURES

1018. LIST OF TABLES

1019. SUMMARY

1020. CONCLUSIONS

1021. RECOMMENDATIONS

1022. REFERENCES

1023. DEFINITIONS

1024. SCOPE

1025. GENERAL PRINCIPLES

1026. TEST METHODS

1027. EVALUATION

1028. REPORTING

1029. APPENDICES

1030. BIBLIOGRAPHY

1031. ANNEXES

1032. INDEX

1033. GLOSSARY

1034. LIST OF FIGURES

1035. LIST OF TABLES

1036. SUMMARY

1037. CONCLUSIONS

1038. RECOMMENDATIONS

1039. REFERENCES

1040. DEFINITIONS

1041. SCOPE

1042. GENERAL PRINCIPLES

1043. TEST METHODS

1044. EVALUATION

1045. REPORTING

1046. APPENDICES

1047. BIBLIOGRAPHY

1048. ANNEXES

1049. INDEX

1050. GLOSSARY

1051. LIST OF FIGURES

1052. LIST

Information technology – Programming languages – MUMPS

Part 1: MUMPS Language Specification

Introduction

Part 1 consists of two sections that describe the MUMPS language. Section 1 describes the metalanguage used in the remainder of Part 1 for the static syntax. Section 2 describes the static syntax and overall semantics of the language. The distinction between "static" and "dynamic" syntax is as follows. The static syntax describes the sequence of characters in a routine as it appears on a tape in routine interchange or on a listing. The dynamic syntax describes the sequence of characters that would be encountered by an interpreter during execution of the routine. (There is no requirement that MUMPS actually be interpreted). The dynamic syntax takes into account transfers of control and values produced by indirection.

1 Static Syntax Metalanguage

The primitives of the metalanguage are the ASCII characters. The metalanguage operators are defined as follows:

Operator	Meaning
::=	definition
[]	option
	grouping
...	optional indefinite repetition
<u>L</u>	list
<u>V</u>	value
<u>SP</u>	space

The following visible representations of ASCII characters required in the defined syntactic objects are used: SP (space), CR (carriage-return), LF (line-feed), and FF (form-feed).

In general, defined syntactic objects will have designators which are underlined names spelled with lower case letters, e.g., name, expr, etc. Concatenation of syntactic objects is expressed by horizontal juxtaposition, choice is expressed by vertical juxtaposition. The ::= symbol denotes a syntactic definition. An optional element is enclosed in square brackets [], and three dots ... denote that the previous element is optionally repeated any number of times. The definition of name, for example, is written:

name ::= | % | [digit / alpha] ...

The vertical bars are used to group elements or to make a choice of elements more readable.

Special care is taken to avoid any danger of confusing the square brackets in the metalanguage with the ASCII graphics] and [. Normally, the square brackets will stand for the metalanguage symbols.

The unary metalanguage operator L denotes a list of one or more occurrences of the syntactic object immediately to its right, with one comma between each pair of occurrences. Thus,

L name is equivalent to name [, name] ...

The binary metalanguage operator V places the constraint on the syntactic object to its left that it must have a value which satisfies the syntax of the syntactic object to its right. For example, one might define the syntax of a hypothetical EXAMPLE command with its argument list by

examplecommand ::= EXAMPLE SP L exampleargument

where

exampleargument ::= | expr / @ expratom V L exampleargument |

This example states: after evaluation of indirection, the command argument list consists of any number of exprs separated by commas. In the static syntax (i.e., prior to evaluation of indirection), occurrences of @ expratom may stand in place of nonoverlapping sublists of command arguments. Usually, the text accompanying a syntax description incorporating indirection will describe the syntax after all occurrences of indirection have been evaluated.

2 Static Syntax and Semantics

2.1 Basic Alphabet

The routine, which is the object whose static syntax is being described in Section 2, is a string made up of the following 98 ASCII symbols.

The 95 printable characters, including the space character represented as SP, and also, the carriage-return character represented as CR, the line-feed character represented as LF, the form-feed character represented as FF.

See 2.4 for the definition of routine.

The syntactic types graphic, alpha, digit, and nonquote are defined here informally in order to save space.

graphic ::= any of the class of 95 ASCII printable characters, including SP.
nonquote ::= any of the characters in graphic except the quote character.
alpha ::= any of the class of 52 upper and lower case letters: A-Z, a-z.
digit ::= any of the class of 10 digits: 0-9.

2.2 Expression Atom exprat

The expression, expr, is the syntactic element which denotes the execution of a value-producing calculation; it is defined in 2.3. The expression atom, exprat, is the basic value-denoting object of which expressions are built; it is defined here.

exprat ::=

<u>lvn</u>
<u>gvn</u>
<u>exprite</u>

See 2.2.2.1 for the definition of lvn. See 2.2.2.2 for the definition of gvn.

exprite ::=

<u>svn</u>
<u>function</u>
<u>exfunc</u>
<u>exvar</u>
<u>numlit</u>
<u>strlit</u>
(<u>expr</u>)
<u>unaryop</u> <u>exprat</u>

See 2.2.6 for the definition of svn. See 2.2.7 for the definition of function. See 2.2.10 for the definition of exfunc. See 2.2.9 for the definition of exvar. See 2.2.3 for the definition of numlit. See 2.2.5 for the definition of strlit. See 2.3 for the definition of expr.

unaryop ::=

'
+
-

 (Note: apostrophe)
(Note: hyphen)

2.2.1 Name name

$$\underline{\text{name}} ::= \left| \begin{array}{c} \% \\ \underline{\text{alpha}} \end{array} \right| \left[\begin{array}{c} \underline{\text{digit}} \\ \underline{\text{alpha}} \end{array} \right] \dots$$

See 2.1 for the definition of alpha and digit.

2.2.2 Variables

The MUMPS standard uses the terms *local variables* and *global variables* somewhat differently from their connotation in certain other computer languages. This section provides a definition of these terms as used in the MUMPS environment.

A MUMPS routine, or set of routines, runs in the context of an operating system process. During its execution, the routine will create and modify variables that are restricted to its process. It can also access (or create) variables that can be shared with other processes. These shared variables will normally be stored on secondary peripheral devices such as disks. At the termination of the process, the process-specific variables cease to exist. The variables created for long term (shared) use remain on auxiliary storage devices where they may be accessed by subsequent processes.

MUMPS uses the term *local variable* to denote variables that are created for use during a single process activation. These variables are not available to other processes. However, they are generally available to all routines executed within the process' lifetime. MUMPS does include certain constructs, the NEW command and parameter passing, which limit the availability of certain variables to specific routines or parts of routines. See 2.2.2.3 for a further discussion of variables and variable environments.

A *global variable* is one that is created by a MUMPS process, but is permanent and shared. As soon as it has been created, it is accessible to other MUMPS processes on the system. Global variables do not disappear when a process terminates. Like local variables, global variables are available to all routines executed within a process.

2.2.2.1 Local Variable Name lvn

$$\underline{\text{lvn}} ::= \left| \begin{array}{c} \underline{\text{rlvn}} \\ @ \underline{\text{expratom}} \vee \underline{\text{lvn}} \end{array} \right|$$

See 2.2 for the definition of expratom. See section 1 for the definition of V.

$$\underline{\text{rlvn}} ::= \left| \begin{array}{c} \underline{\text{name}} [(\underline{\text{L}} \underline{\text{expr}})] \\ @ \underline{\text{lnamind}} @ (\underline{\text{L}} \underline{\text{expr}}) \end{array} \right|$$

See 2.2.1 for the definition of name. See 2.3 for the definition of expr. See section 1 for the definition of L.

$$\underline{\text{lnamind}} ::= \underline{\text{rexpratom}} \vee \underline{\text{lvn}}$$

See section 1 for the definition of V.

$$\underline{\text{rexpratom}} ::= \left| \begin{array}{c} \underline{\text{rlvn}} \\ \underline{\text{rqvn}} \\ \underline{\text{expritem}} \end{array} \right|$$

See 2.2.2.2 for the definition of rgvn. See 2.2 for the definition of exprimem.

A local variable name is either unsubscripted or subscripted; if it is subscripted, any number of subscripts separated by commas is permitted. An unsubscripted occurrence of lvn may carry a different value from any subscripted occurrence of lvn.

When lnamind is present it is always a component of an rlvn. If the value of the rlvn is a subscripted form of lvn, then some of its subscripts may have originated in the lnamind. In this case, the subscripts contributed by the lnamind appear as the first subscripts in the value of the resulting rlvn, separated by a comma from the (non-empty) list of subscripts appearing in the rest of the rlvn.

2.2.2.2 Global Variable Name gvn

$$\underline{gvn} ::= \left| \begin{array}{l} \underline{rgvn} \\ @ \underline{expratom} \vee \underline{gvn} \end{array} \right|$$

See 2.2 for the definition of expratom. See section 1 for the definition of V.

$$\underline{rgvn} ::= \left| \begin{array}{l} ^{(\underline{L} \underline{expr})} \\ ^{\underline{name} [(\underline{L} \underline{expr})]} \\ @ \underline{gnamind} @ (\underline{L} \underline{expr}) \end{array} \right|$$

See 2.3 for the definition of expr. See 2.2.1 for the definition of name. See section 1 for the definition of L.

$$\underline{gnamind} ::= \left| \underline{rexpratom} \vee \underline{gvn} \right|$$

See section 1 for the definition of V.

The prefix $\wedge$ uniquely denotes a global variable name. A global variable name is either unsubscripted or subscripted; if it is subscripted, any number of subscripts separated by commas is permitted. An abbreviated form of subscripted gvn is permitted, called the *naked reference*, in which the prefix is present but the name and an initial (possibly empty) sequence of subscripts is absent but implied by the value of the *naked indicator*. An unsubscripted occurrence of gvn may carry a different value from any subscripted occurrence of gvn.

When gnamind is present it is always a component of an rgvn. If the value of the rgvn is a subscripted form of gvn, then some of its subscripts may have originated in the gnamind. In this case, the subscripts contributed by the gnamind appear as the first subscripts in the value of the resulting rgvn, separated by a comma from the (non-empty) list of subscripts appearing in the rest of the rgvn.

Every executed occurrence of gvn affects the naked indicator as follows. If, for any positive integer m , the gvn has the nonnaked form

$$N(v_1, v_2, \dots, v_m)$$

then the m -tuple $N, v_1, v_2, \dots, v_{m-1}$, is placed into the naked indicator when the gvn reference is made. A subsequent naked reference of the form

$$\wedge(s_1, s_2, \dots, s_i) \quad (i \text{ positive})$$

results in a global reference of the form

$$N(v_1, v_2, \dots, v_{m-1}, s_1, s_2, \dots, s_i)$$

after which the $m+i-1$ -tuple $N, v_1, v_2, \dots, s_{i-1}$ is placed into the naked indicator. Prior to the first executed occurrence of a nonnaked form of gvn, the value of the naked indicator is undefined. It is erroneous for the first executed occurrence of gvn to be a naked reference. A nonnaked reference without subscripts leaves the naked indicator undefined.

The effect on the naked indicator described above occurs regardless of the context in which gvn is found; in particular, an assignment of a value to a global variable with the command SET gvn = expr does not affect the value of the naked indicator until after the right-side expr has been evaluated. The effect on the naked indicator of any gvn within the right-side expr will precede the effect on the naked indicator of the left-side gvn.

For convenience, glvn is defined so as to be satisfied by the syntax of either gvn or lvn.

$$\underline{glvn} ::= \left| \begin{array}{c} \underline{gvn} \\ \underline{lvn} \end{array} \right|$$

See 2.2.2.1 for the definition of lvn.

2.2.2.3 Variable Handling

MUMPS has no explicit declaration or definition statements. Local and global variables, both non-subscripted and subscripted, are automatically created as data is stored into them, and their data contents can be referred to once information has been stored. Since the language has only one data type - string - there is no need for type declarations or explicit data type conversions. Array structures can be multidimensional with data simultaneously stored at all levels including the variable name level. Subscripts can be positive, negative, and/or noninteger numbers as well as nonnumeric strings (other than empty strings).

In general, the operation of the local variable symbol table can be viewed as follows. Prior to the initial setting of information into a variable, the data value of that variable is said to be undefined. Data is stored into a variable with commands such as SET, READ, or FOR. Subsequent references to that variable return the data value that was most recently stored. When a variable is killed, as with the KILL command, that variable and all of its array descendants (if any) are deleted, and their data values become undefined.

No explicit syntax is needed for a routine or subroutine to have access to the local variables of its caller. Except when the NEW command or parameter passing is being used, a subroutine or called routine (the callee) has the same set of variable values as its caller and, upon completion of the called routine or subroutine, the caller resumes execution with the same set of variable values as the callee had at its completion.

The NEW command provides scoping of local variables. It causes the current values of a specified set of variables to be saved. The variables are then set to undefined data values. Upon returning to the caller of the current routine or subroutine, the saved values, including any undefined states, are restored to those variables. Parameter passing, including the DO command, extrinsic functions, and extrinsic variables, allows parameters to be passed into a subroutine or routine without the callee being concerned with the variable names used by the caller for the data being passed or returned.

The formal association of MUMPS local variables with their values can best be described by a conceptual model. This model is NOT meant to imply an implementation technique for a MUMPS processor.

The value of a MUMPS variable may be described by a relationship between two structures: the NAME-TABLE and the VALUE-TABLE. (In reality, at least two such table sets are required, one pair per executing process for process-specific local variables and one pair for system-wide global variables.) Since the value association process is the same for both types of variables, and since issues of scoping due to parameter

passing or nested environments apply only to local variables, the discussion that follows will address only local variable value association. It should be noted, however, that while the overall structures of the table sets are the same, there are two major differences in the way the sets are used. First, the global variable tables are shared. This means that any operations on the global tables, e.g., SET or KILL, by one process, affect the tables for all processes. Second, since scoping issues of parameter passing and the NEW command are not applicable to global variables, there is always a one-to-one relationship between entries in the global NAME-TABLE (variable names) and entries in the global VALUE-TABLE (values).

The NAME-TABLE consists of a set of entries, each of which contains a name and a pointer. This pointer represents a correspondence between that name and exactly one DATA-CELL from the VALUE-TABLE. The VALUE-TABLE consists of a set of DATA-CELLs, each of which contains zero or more tuples of varying degrees. The degree of a tuple is the number (possibly 0) of elements or subscripts in the tuple list. Each tuple present in the DATA-CELL has an associated data value.

The NAME-TABLE entries contain every non-subscripted variable or array name (name) known, or accessible, by the MUMPS process in the current environment. The VALUE-TABLE DATA-CELLs contain the set of tuples that represent all variables currently having data-values for the process. Every name (entry) in the NAME-TABLE refers (points) to exactly one DATA-CELL, and every entry contains a unique name. Several NAME-TABLE entries (names) can refer to the same DATA-CELL, however, and thus there is a many-to-one relationship between (all) NAME-TABLE entries and DATA-CELLs. A name is said to be *bound* to its corresponding DATA-CELL through the pointer in the NAME-TABLE entry. Thus the pointer is used to represent the correspondence and the phrase *change the pointer* is the equivalent to saying *change the correspondence so that a name now corresponds to a possible different DATA-CELL (value)*. NAME-TABLE entries are also placed in the PROCESS-STACK (see 2.2.2.4).

The value of an unsubscripted lvn corresponds to the tuple of degree 0 found in the DATA-CELL that is bound to the NAME-TABLE entry containing the name of the lvn. The value of a subscripted lvn (array node) of degree *n* also corresponds to a tuple in the DATA-CELL that is bound to the NAME-TABLE entry containing the name of the lvn. The specific tuple in that DATA-CELL is the tuple of degree *n* such that each subscript of the lvn has the same value as the corresponding element of the tuple. If the designated tuple doesn't exist in the DATA-CELL then the corresponding lvn is said to be *undefined*.

In the following figure, the variables and array nodes have the designated data values.

```
VAR1 = "Hello"
VAR2 = 12.34
VAR3 = "abc"
VAR3("Smith","John",1234)=123
VAR3("Widget","red") = -56
```

Also, the variable *DEF* existed at one time but no longer has any data or array value, and the variable *XYZ* has been *bound* through parameter passing to the same data and array information as the variable *VAR2*.

NAME-TABLE	VALUE-TABLE DATA-CELLS
VAR1----->	()="Hello"
VAR2----->	()=12.34
XYZ----->	
VAR3----->	()="abc" ("Smith","John",1234)=123 ("Widget","red")=-56
DEF----->	

The initial state of a MUMPS process prior to execution of any MUMPS code consists of an empty NAME-TABLE and VALUE-TABLE. When information is to be stored (*set*, *given*, or *assigned*) into a variable (l_{vn}):

- If the name of the l_{vn} does not already appear in an entry in the NAME-TABLE, an entry is added to the NAME-TABLE which contains the name and a pointer to a new (empty) DATA-CELL. The corresponding DATA-CELL is added to the VALUE-TABLE without any initial tuples.
- Otherwise, the pointer in the NAME-TABLE entry which contained the name of the l_{vn} is extracted. The operations in step c. and d. refer to tuples in that DATA-CELL referred to by this pointer.
- If the l_{vn} is unsubscripted, then the tuple of degree 0 in the DATA-CELL has its data value replaced by the new data value. If that tuple did not already exist, it is created with the new data value.
- If the l_{vn} is subscripted, then the tuple of subscripts in the DATA-CELL (i.e., the tuple created by dropping the name of the l_{vn}; the degree of the tuple equals the number of subscripts) has its data value replaced by the new data value. If that tuple did not already exist, it is created with the new data value.

When information is to be retrieved, if the name of the l_{vn} is not found in the NAME-TABLE, or if its corresponding DATA-CELL tuple does not exist, then the data value is said to be undefined. Otherwise, the data value exists and is retrieved. A data value of the empty string (a string of zero length) is not the same as an undefined data value.

When a variable is deleted (*killed*):

- a. If the name of the lvn is not found in the NAME-TABLE, no further action is taken.
- b. If the lvn is unsubscripted, all of the tuples in the corresponding DATA-CELL are deleted.
- c. If the lvn is subscripted, let N be the degree of the subscript tuple formed by removing the name from the lvn. All tuples that satisfy the following two conditions are deleted from the corresponding DATA-CELL:
 1. The degree of the tuple must be greater than or equal to N , and
 2. The first N arguments of the tuple must equal the corresponding subscripts of the lvn.

In this formal language model, even if all of the tuples in a DATA-CELL are deleted, neither the DATA-CELL nor the corresponding names in the NAME-TABLE are ever deleted. Their continued existence is frequently required as a result of parameter passing and the NEW command.

2.2.2.4 Variable Contexts

The organization of multiple variable contexts requires the use of a PROCESS-STACK. This is a simple push-down stack, or last-in-first-out (LIFO) list, used to save and restore items which control the execution flow or variable environment. Five types of items, or frames, will be placed on the PROCESS-STACK, DO frames, exfunc frames, exvar frames, NEW frames, and parameter frames:

- a. DO frames contain the execution level and the execution location of the doargument. In the case of the argumentless DO, the execution level, the execution location of the DO command and a saved value of \$T are saved. The execution location of a MUMPS process is a descriptor of the location of the command and possible argument currently being executed. This descriptor includes, at minimum, the routinename and the character position following the current command or argument.
- b. Exfunc and exvar frames contain saved values of \$T, the execution level, and the execution location.
- c. NEW frames contain a NEW argument (newargument) and a set of NAME-TABLE entries.
- d. Parameter frames contain a formallist and a set of NAME-TABLE entries.

2.2.3 Numeric Literal numlit

The integer literal syntax, intl, which is a nonempty string of digits, is defined here.

intl ::= digit ...

See 2.1 for the definition of digit.

The numeric literal numlit is defined as follows.

numlit ::= mant [exp]

$$\underline{\text{mant}} ::= | \underline{\text{intlit}} [. \underline{\text{intlit}}] |$$

$$\underline{\text{exp}} ::= E \begin{bmatrix} + \\ - \end{bmatrix} \underline{\text{intlit}}$$

The value of the string denoted by an occurrence of numlit is defined in the following two subsections.

2.2.3.1 Numeric Data Values

All variables, local, global, and special, have values which are either defined or undefined. If defined, the values may always be thought of and operated upon as strings. The set of numeric values is a subset of the set of all data values.

Only numbers which may be represented with a finite number of decimal digits are representable as numeric values. A data value has the form of a number if it satisfies the following restrictions.

- a. It may contain only digits and the characters "-" and ".".
- b. At least one digit must be present.
- c. "." occurs at most once.
- d. The number zero is represented by the one-character string "0".
- e. The representation of each positive number contains no "-".
- f. The representation of each negative number contains the character "-" followed by the representation of the positive number which is the absolute value of the negative number. (Thus, the following restrictions describe positive numbers only.)
- g. The representation of each positive integer contains only digits and no leading zero.
- h. The representation of each positive number less than 1 consists of a "." followed by a nonempty digit string with no trailing zero. (This is called a *fraction*.)
- i. The representation of each positive noninteger greater than 1 consists of the representation of a positive integer (called the *integer part* of the number) followed by a fraction (called the *fraction part* of the number).

Note that the mapping between representable numbers and representations is one-to-one. An important result of this is that string equality of numeric values is a necessary and sufficient condition of numeric equality.

2.2.3.2 Meaning of numlit

Note that numlit denotes only nonnegative values. The process of converting the spelling of an occurrence of numlit into its numeric data value consists of the following steps.

- a. If the mant has no ".", place one at its right end.
- b. If the exp is absent, skip step c.
- c. If the exp has a plus or has no sign, move the "." a number of decimal digit positions to the right in the mant equal to the value of the intlit of exp, appending zeros to the right of the mant as necessary. If the exp has a minus sign, move the "." a number of decimal digit positions to the left in the mant equal to the value of the intlit of exp, appending zeros to the left of the mant as necessary.
- d. Delete the exp and any leading or trailing zeros of the mant.
- e. If the rightmost character is ".", remove it.
- f. If the result is empty, make it "0".

2.2.4 Numeric Interpretation of Data

Certain operations, such as arithmetic, deal with the numeric interpretations of their operands. The numeric interpretation is a mapping from the set of all data values into the set of all numeric values, described by the following algorithm. Note that the numeric interpretation maps numeric values into themselves.

(Note: The *head* of a string is defined to be a substring which contains an identical sequence of characters in the string to the left of a given point and none of the characters in the string to the right of that point. A head may be empty or it may be the entire string.)

Consider the argument to be the string *S*.

First, apply the following sign reduction rules to *S* as many times as possible, in any order.

- a. If *S* is of the form $+ T$, then remove the $+$. (Shorthand: $+ T \rightarrow T$)
- b. $- + T \rightarrow - T$
- c. $- - T \rightarrow T$

Second, apply one of the following, as appropriate.

- a. If the leftmost character of *S* is not "-", form the longest head of *S* which satisfies the syntax description of numlit. Then apply the algorithm of 2.2.3.2 to the result.
- b. If *S* is of the form $- T$, apply step a. above to *T* and append a "-" to the left of the result. If the result is "-0", change it to "0".

The *numeric expression* numexpr is defined to have the same syntax as expr. Its presence in a syntax description serves to indicate that the numeric interpretation of its value is to be taken when it is executed.

numexpr ::= expr

See 2.3 for the definition of expr.

2.2.4.1 Integer Interpretation

Certain functions deal with the integer interpretations of their arguments. The integer interpretation is a mapping from the set of all data values onto the set of all integer values, described by the following algorithm.

First, take the numeric interpretation of the argument. Then remove the fraction, if present. If the result is empty or "-", change it to "0".

The *integer expression* intexpr is defined to have the same syntax as expr. Its presence in a syntax definition serves to indicate that the integer interpretation of its value is to be taken when it is executed.

intexpr ::= expr

See 2.3 for the definition of expr.

2.2.4.2 Truth-Value Interpretation

The truth-value interpretation is a mapping from the set of all data values onto the two integer values 0 (false) and 1 (true), described by the following algorithm. Take the numeric interpretation. If the result is not "0", make it "1".

The *truth-value expression* tvexpr is defined to have the same syntax as expr. Its presence in a syntax definition serves to indicate that the truth-value interpretation of its value is to be taken when it is executed.

tvexpr ::= expr

See 2.3 for the definition of expr.

2.2.5 String Literal strlit

strlit ::= " " "
nongquote ... "

See 2.1 for the definition of nongquote.

In words, a string literal is bounded by quotes and contains any string of printable characters, except that when quotes occur inside the string literal, they occur in adjacent pairs. Each such adjacent quote pair denotes a single quote in the value denoted by strlit, whereas any other printable character between the bounding quotes denotes itself. An empty string is denoted by exactly two quotes.

2.2.6 Intrinsic Special Variable Name svn

Intrinsic special variables are denoted by the prefix \$ followed by one of a designated list of names. Intrinsic special variable names differing only in the use of corresponding upper and lower case letters are equivalent. The standard contains the following intrinsic special variable names:

H[OROLOG]
I[O]
J[OB]
S[TORAGE]
T[EST]
X
Y
Z[unspecified]

Unused intrinsic special variable names beginning with an initial letter other than Z are reserved for future enhancement of the standard.

The formal definition of the syntax of svn is a choice from among all of the individual svn syntax definitions of 2.2.6.

<u>svn</u> ::=		syntax of \$HOROLOG intrinsic special variable	
		syntax of \$IO intrinsic special variable	
		.	
		syntax of \$Y intrinsic special variable	

Any implementation of the language must be able to recognize both the abbreviation and the full spelling of each intrinsic special variable name.

<u>Syntax</u>	<u>Definition</u>
\$H[OROLOG]	\$H gives date and time with one access. Its value is <i>D</i> , <i>S</i> where <i>D</i> is an integer value counting days since an origin specified below, and <i>S</i> is an integer value modulo 86,400 counting seconds. The value of \$H for the first second of December 31, 1840 is defined to be 0,0. <i>S</i> increases by 1 each second and <i>S</i> clears to 0 with a carry into <i>D</i> on the tick of midnight.
\$I[O]	\$I identifies the current I/O device (see 2.6.2 and 2.6.18).
\$J[OB]	Each executing MUMPS process has its own job number, a positive integer which is the value of \$J. The job number of each process is unique to that process within a domain of concurrent processes defined by the implementor. \$J is constant throughout the active life of a process.
\$S[TORAGE]	Each implementation must return for the value of \$S an integer which is the number of characters of free space available for use. The method of arriving at the value of \$S is not part of the standard.
\$T[EST]	\$T contains the truth value computed from the execution of an IF <u>command</u> containing an argument, or an OPEN, LOCK, JOB, or READ <u>command</u> with a timeout (see 2.2.9, 2.2.10, and 2.6.3).
\$X	\$X has a nonnegative integer value which approximates the value of a carriage or horizontal cursor position on the current line as if the current I/O device were an

ASCII terminal. It is initialized to zero by input or output of control functions corresponding to CR or FF; input or output of each graphic adds 1 to \$X (see 2.5.5 and 2.6.18).

\$Y \$Y has a nonnegative integer value which approximates the line number on the current I/O device as if it were an ASCII terminal. It is initialized to zero by input or output of control functions corresponding to FF; input or output of control functions corresponding to LF adds 1 to \$Y (see 2.5.5 and 2.6.18).

\$Z[unspecified] Z is the initial letter reserved for defining non-standard intrinsic special variables. The requirement that \$Z be used permits the unused initial letters to be reserved for future enhancement of the standard without altering the execution of existing routines which observe the rules of the standard.

2.2.7 Intrinsic Functions function

Intrinsic functions are denoted by the prefix \$ followed by one of a designated list of names, followed by a parenthesized argument list. Intrinsic function names differing only in the use of corresponding upper and lower case letters are equivalent. The standard contains the following function names:

```
A[SCII]
C[HAR]
D[ATA]
E[XTRACT]
F[IND]
FN[UMBER]
G[ET]
J[USTIFY]
L[ENGTH]
N[EXT]
O[RDER]
P[IECE]
Q[UEY]
R[ANDOM]
S[ELECT]
T[EXT]
TR[ANSLATE]
V[IEW]
Z[unspecified]
```

Unused function names beginning with an initial letter other than Z are reserved for future enhancement of the standard.

The formal definition of the syntax of function is a choice from among all of the individual function syntax definitions of 2.2.7

<u>function</u> ::=	syntax of \$ASCII function
	syntax of \$CHAR function
	.
	syntax of \$VIEW function

Any implementation of the language must be able to recognize both the abbreviation and the full spelling of each function name.

2.2.7.1 \$ASCII

\$A[SCII] (expr)

See 2.3 for the definition of expr.

This form produces an integer value as follows:

- a. -1 if the value of expr is the empty string.
- b. Otherwise, an integer n associated with the leftmost character of the value of expr, such that $\$A(\$C(n)) = n$.

$\$A[SCII] (\text{expr} , \text{intexpr})$

See 2.3 for the definition of expr. See 2.2.4.1 for the definition of intexpr.

This form is similar to $\$A(\text{expr})$ except that it works with the intexprth character of expr instead of the first. Formally, $\$A(\text{expr}, \text{intexpr})$ is defined to be $\$A(\$E(\text{expr}, \text{intexpr}))$.

2.2.7.2 \$CHAR

$\$C[HAR] (\underline{L} \text{intexpr})$

See 2.2.4.1 for the definition of intexpr. See section 1 for the definition of L.

This form returns a string whose length is the number of argument expressions which have nonnegative values. Each intexpr in the closed interval $[0, 127]$ maps into the ASCII character whose code is the value of intexpr; this mapping is order-preserving. Each negative-valued intexpr maps into no character in the value of $\$C$.

2.2.7.3 \$DATA

$\$D[ATA] (\underline{glvn})$

See 2.2.2.2 for the definition of glvn.

This form returns a nonnegative integer which is a characterization of the glvn. The value of the integer is $p+d$, where:

$d = 1$ if the glvn has a defined value, i.e., the NAME-TABLE entry for the name of the glvn exists, and the subscript tuple of the glvn has a corresponding entry in the associated DATA-CELL; otherwise, $d=0$.

$p = 10$ if the variable has descendants; i.e., there exists at least one tuple in the glvn's DATA-CELL which satisfies the following conditions:

- a. The degree of the tuple is greater than the degree of the glvn, and
- b. the first N arguments of the tuple are equal to the corresponding subscripts of the glvn where N is the number of subscripts in the glvn.

If no NAME-TABLE entry for the glvn exists, or no such tuple exists in the associated DATA-CELL, then $p=0$.

2.2.7.4 \$EXTRACT**\$E[XTRACT]** (expr)See 2.3 for the definition of expr.

This form returns the first (leftmost) character of the value of expr. If the value of expr is the empty string, the empty string is returned.

\$E[XTRACT] (expr , intexpr)See 2.3 for the definition of expr. See 2.2.4.1 for the definition of intexpr.

Let s be the value of expr, and let m be the integer value of intexpr. $\$E(s,m)$ returns the m th character of s . If m is less than 1 or greater than $\$L(s)$, the value of $\$E$ is the empty string. (1 corresponds to the leftmost character of s ; $\$L(s)$ corresponds to the rightmost character.)

\$E[XTRACT] (expr , intexpr₁ , intexpr₂)See 2.3 for the definition of expr. See 2.2.4.1 for the definition of intexpr.

Let n be the integer value of intexpr₂. $\$E(s,m,n)$ returns the string between positions m and n of s . The following cases are defined:

- a. $m > n$. Then the value of $\$E$ is the empty string.
- b. $m = n$. $\$E(s,m,n) = \$E(s,m)$.
- c. $m < n \leq \$L(s)$.
 $\$E(s,m,n) = \$E(s,m)$ concatenated with $\$E(s,m+1,n)$.
 That is, using the concatenation operator $_$ of 2.3.5, $\$E(s,m,n) = \$E(s,m)_ \$E(s,m+1)_ \dots _ \$E(s,m+(n-m))$.
- d. $m < n$ and $\$L(s) < n$.
 $\$E(s,m,n) = \$E(s,m,\$L(s))$.

2.2.7.5 \$FIND**\$F[IND]** (expr₁ , expr₂)See 2.3 for the definition of expr.

This form searches for the leftmost occurrence of the value of expr₂ in the value of expr₁. If none is found, $\$F$ returns zero. If one is found, the value returned is the integer representing the number of the character position immediately to the right of the rightmost character of the found occurrence of expr₂ in expr₁. In particular, if the value of expr₂ is empty, $\$F$ returns 1.

\$F[IND] (expr₁ , expr₂ , intexpr)See 2.3 for the definition of expr. See 2.2.4.1 for the definition of intexpr.

Let a be the value of expr_1 , let b be the value of expr_2 , and let m be the value of intexpr . $\$F(a,b,m)$ searches for the leftmost occurrence of b in a , beginning the search at the $\max(m,1)$ position of a . Let p be the value of the result of $\$F(\$E(a,m,\$L(a)),b)$. If no instance of b is found (i.e., $p=0$), $\$F$ returns the value 0; otherwise, $\$F(a,b,m) = p + \max(m,1) - 1$.

2.2.7.6 \$FNUMBER

\$FN[UMBER] (numexpr , fncodeexpr)

See 2.2.4 for the definition of numexpr.

fncodeexpr ::= expr V fncode

See 2.3 for the definition of expr. See section 1 for the definition of V.

fncode ::= fncodatom ...

fncodatom ::=

P	
T	
,	(note, comma)
+	
-	(note, hyphen)

returns a value which is an edited form of numexpr. Each fncodatom is applied to numexpr in formatting the results by the following rules (order of processing is not significant):

<u>fncodatom</u>	Action
P	Represent negative <u>numexpr</u> values in parentheses. Let A be the absolute value of <u>numexpr</u> . Use of <u>fncode</u> "P" will result in the following: 1. If <u>numexpr</u> < 0, the result will be "(" <u>A</u> ")". 2. If <u>numexpr</u> > 0, the result will be " " <u>A</u> " ".
T	Represent <u>numexpr</u> with a trailing rather than a leading "+" or "-" sign. Note: if sign suppression is in force (either by default on positive values, or by design using the "-" <u>fncodatom</u>), use of <u>fncode</u> "T" will result in a trailing space character.
,	Insert comma delimiters every third position to the left of the decimal (present or assumed) within <u>numexpr</u> . Note: no comma shall be inserted which would result in a leading comma character.
+	Force a plus sign ("+") on positive values of <u>numexpr</u> . Position of the "+" (leading or trailing) is dependent on whether or not <u>fncodatom</u> of "T" is specified.
-	Suppress the negative sign "-" on negative values of <u>numexpr</u> .

If fncodeexpr equals an empty string, no special formatting is performed and the result of the expression is the original value of numexpr.

More than one occurrence of a particular fncodatom within a single fncode is identical to a single occurrence of that fncodatom. Erroneous conditions are produced when a fncodatom "P" is present with any of the sign suppression or sign placement fncodatoms ("T+-").

\$FN[UMBER] (numexpr , fncoexpr , intexpr)

See 2.2.4 for the definition of numexpr. See 2.2.4.1 for the definition of intexpr.

This form is identical to the two-argument form of \$FN, except that numexpr is rounded to intexpr fraction digits, including possible trailing zeros, before processing any fncoatoms. If intexpr is zero, the evaluated numexpr contains no decimal point. Note: if $(-1 < \text{numexpr} < 1)$, the result of \$FN has a leading zero ("0") to the left of the decimal point.

2.2.7.7 \$GET

\$G[ET] (glvn)

See 2.2.2.2 for the definition of glvn.

This form returns the value of the specified glvn depending on its state, defined by \$D(glvn). The following cases are defined:

- a. $\$D(\text{glvn})\#10 = 1$
The value returned is the value of the variable specified by glvn.
- b. Otherwise, the value returned is the empty string.

2.2.7.8 \$JUSTIFY

\$J[USTIFY] (expr , intexpr)

See 2.3 for the definition of expr. See 2.2.4.1 for the definition of intexpr.

This form returns the value of expr right-justified in a field of intexpr spaces. Let m be $\$L(\text{expr})$ and n be the value of intexpr. The following cases are defined:

- a. $m' < n$. Then the value returned is expr.
- b. Otherwise, the value returned is $S(n-m)$ concatenated with expr, where $S(x)$ is a string of x spaces.

\$J[USTIFY] (numexpr , intexpr₁ , intexpr₂)

See 2.2.4 for the definition of numexpr. See 2.2.4.1 for the definition of intexpr.

This form returns an edited form of the number numexpr. Let r be the value of numexpr after rounding to intexpr₂ fraction digits, including possible trailing zeros. (If intexpr₂ is the value 0, r contains no decimal point.) The value returned is $\$J(r, \text{intexpr}_1)$. Note that if $-1 < \text{numexpr} < 1$, the result of \$J does have a zero to the left of the decimal point. Negative values of intexpr₂ are reserved for future extensions of the \$JUSTIFY function.

2.2.7.9 \$LENGTH

\$L[ENGTH] (expr)

See 2.3 for the definition of expr.

This form returns an integer which is the number of characters in the value of expr. If the value of expr is the empty string, \$L(expr) returns the value 0.

\$L[ENGTH] (expr₁ , expr₂)

See 2.3 for the definition of expr.

This form returns the number plus one of nonoverlapping occurrences of expr₂ in expr₁. If the value of expr₂ is the empty string, then \$L returns the value 0.

2.2.7.10 \$NEXT¹

\$N[EXT] (glvn)

See 2.2.2.2 for the definition of glvn.

This form is included for backward compatibility. The use of the \$ORDER function is strongly encouraged in place of \$NEXT, as the two functions perform the same operation except for the different starting and ending condition of \$NEXT.

\$N returns a value which is a subscript according to a subscript ordering sequence. This ordering sequence is specified below with the aid of a function, CO, which is used for definitional purposes only, to establish the collating sequence.

CO(*s*,*t*) is defined, for strings *s* and *t*, as follows:

When *t* follows *s* in the ordering sequence, CO(*s*,*t*) returns *t*.
Otherwise, CO(*s*,*t*) returns *s*.

Let *m* and *n* be strings satisfying the definition of numeric data values (see 2.2.3.1), and *u* and *v* be nonempty strings which do not satisfy this definition. The following cases define the ordering sequence:

- a. CO("",*s*) = *s*.
- b. CO(*m*,*n*) = *n* if *n* > *m*; otherwise, CO(*m*,*n*) = *m*.
- c. CO(*m*,*u*) = *u*.
- d. CO(*u*,*v*) = *v* if *v*] *u*; otherwise, CO(*u*,*v*) = *u*.

In words, all strings follow the empty string, numerics collate in numeric order, numerics precede nonnumeric strings, and nonnumeric strings are ordered by the conventional ASCII collating sequence.

Only subscripted forms of lvn and gvn are permitted. Let lvn or gvn be of the form Name(*s*₁, *s*₂, ..., *s*_{*n*}). If *s*_{*n*} is -1, let *A* be the set of all subscripts. If *s*_{*n*} is not -1, let *A* be the set of all subscripts that follow *s*_{*n*}; that is, for all *s* in *A*:

¹\$NEXT may not be included in the next version of ISO/IEC 11756.

- a. $CO(s_n, s) = s$ and
- b. $\$D(\text{Name}(s_1, s_2, \dots, s_{n-1}, s))$ is not zero.

Then $\$N(\text{Name}(s_1, s_2, \dots, s_n))$ returns that value t in A such that $CO(t, s) = s$ for all s not equal to t ; that is, all other subscripts which follow s_n also follow t .

If no such t exists, -1 is returned.

Note that $\$N$ will return ambiguous results for lvn and gvn arrays which have negative numeric subscript values.

2.2.7.11 \$ORDER

$\$O[RDER]$ (glvn)

See 2.2.2.2 for the definition of glvn.

This form returns a value which is a subscript according to a subscript ordering sequence. This ordering sequence is specified below with the aid of a function, CO , which is used for definitional purposes only, to establish the collating sequence.

$CO(s, t)$ is defined, for strings s and t , as follows:

- When t follows s in the ordering sequence, $CO(s, t)$ returns t .
- Otherwise, $CO(s, t)$ returns s .

Let m and n be strings satisfying the definition of numeric data values (see 2.2.3.1), and u and v be nonempty strings which do not satisfy this definition. The following cases define the ordering sequence:

- a. $CO("", s) = s$.
- b. $CO(m, n) = n$ if $n > m$; otherwise, $CO(m, n) = m$.
- c. $CO(m, u) = u$.
- d. $CO(u, v) = v$ if $v \succ u$; otherwise, $CO(u, v) = u$.

In words, all strings follow the empty string, numerics collate in numeric order, numerics precede nonnumeric strings, and nonnumeric strings are ordered by the conventional ASCII collating sequence.

Only subscripted forms of lvn and gvn are permitted. Let lvn or gvn be of the form $\text{Name}(s_1, s_2, \dots, s_n)$ where s_n may be the empty string. Let A be the set of subscripts that follow s_n . That is, for all s in A :

- a. $CO(s_n, s) = s$ and
- b. $\$D(\text{Name}(s_1, s_2, \dots, s_{n-1}, s))$ is not zero.

Then $\$O(\text{Name}(s_1, s_2, \dots, s_n))$ returns that value t in A such that $CO(t, s) = s$ for all s not equal to t ; that is, all other subscripts which follow s_n also follow t .

If no such t exists, $\$O$ returns the empty string.

2.2.7.12 \$PIECE

$\$P[IECE]$ (expr₁ , expr₂)

See 2.3 for the definition of expr.

This form is defined here with the aid of a function, NF, which is used for definitional purposes only, called *find the position number following the mth occurrence*.

NF(s, d, m) is defined, for strings s , d , and integer m , as follows:

When d is the empty string, the result is zero.

When $m > 0$, the result is zero.

When d is not a substring of s , i.e., when $\$F(s, d) = 0$, then the result is $\$L(s) + \$L(d) + 1$.

Otherwise, $NF(s, d, 1) = \$F(s, d)$.

For $m > 1$, $NF(s, d, m) = NF(\$E(s, \$F(s, d), \$L(s)), d, m-1) + \$F(s, d) - 1$.

That is, NF extends $\$F$ to give the position number of the character to the right of the m th occurrence of the string d in s .

Let s be the value of expr₁, and let d be the value of expr₂. $\$P(s, d)$ returns the substring of s bounded on the right but not including the first (leftmost) occurrence of d .

$\$P(s, d) = \$E(s, 0, NF(s, d, 1) - \$L(d) - 1)$.

$\$P[IECE] (\text{expr}_1, \text{expr}_2, \text{intexpr})$

See 2.3 for the definition of expr. See 2.2.4.1 for the definition of intexpr.

Let m be the integer value of intexpr. $\$P(s, d, m)$ returns the substring of s bounded by but not including the $m-1$ th and the m th occurrence of d .

$\$P(s, d, m) = \$E(s, NF(s, d, m-1), NF(s, d, m) - \$L(d) - 1)$.

$\$P[IECE] (\text{expr}_1, \text{expr}_2, \text{intexpr}_1, \text{intexpr}_2)$

See 2.3 for the definition of expr. See 2.2.4.1 for the definition of intexpr.

Let n be the integer value of intexpr₂. $\$P(s, d, m, n)$ returns the substring of s bounded on the left but not including the $m-1$ th occurrence of d in s , and bounded on the right but not including the n th occurrence of d in s .

$\$P(s, d, m, n) = \$E(s, NF(s, d, m-1), NF(s, d, n) - \$L(d) - 1)$.

Note that $\$P(s, d, m, m) = \$P(s, d, m)$, and that $\$P(s, d, 1) = \$P(s, d)$.

2.2.7.13 \$QUERY

$\$Q[QUERY] (\text{glvn})$

See 2.2.2.2 for the definition of glvn.

Let $\text{glvn}_1 = \text{A}(i_1, i_2, \dots, i_p)$ and $\text{glvn}_2 = \text{A}(j_1, j_2, \dots, j_q)$

Then a collating relation exists between glvn₁ and glvn₂. Glvn₂ is said to follow glvn₁ when for any pair (i_k, j_k) when:

- a. $p < q$ and $i_k = j_k$ for all k in the range $(0 < k' > p)$.
- or
- b. $p = 0$ and $q > 0$.
- or
- c. $k > 0$ and $k' > \min(p, q)$ and
 there exists some $i_k' = j_k$ and
 no n exists with $0 < n < k$ and $i_n' = j_n$ and
 the function $\text{CO}(i_k, j_k)$, as defined in 2.2.7 (definition of \$ORDER), is equal to j_k .
 In less formal terms, when the first index that is different, collates in glvn₂ after the corresponding one in glvn₁.

For the purpose of this discussion a function $\text{CQ}(\text{glvn}_1, \text{glvn}_2)$ is defined that would yield glvn₂ when, according to the above definition, glvn₂ would be said to follow glvn₁.

namevalue ::= expr

See 2.3 for the definition of expr.

A namevalue has the syntax of a glvn with the following restrictions:

- a. The glvn is not a naked reference.
- b. Each subscript whose value has the form of a number as defined in 2.2.3.1 appears as a numlit, spelled as its numeric interpretation.
- c. Each subscript whose value does not have the form of a number as defined in 2.2.3.1 appears as a sublit, defined as follows:

sublit ::= " | " " | subnonquote | ... "

where subnonquote is defined as follows:

subnonquote ::= any character valid in a subscript, excluding the quote symbol.

Then the value of the function \$QUERY can be defined as follows:

the value of $\text{\$QUERY}(\text{glvn}_1)$ is a namevalue that conforms to glvn₂ if and only if:

$\text{CQ}(\text{glvn}_1, \text{glvn}_2) = \text{glvn}_2$

and no glvn₃ exists so that

$\text{CQ}(\text{glvn}_1, \text{glvn}_3) = \text{glvn}_3$

and

$\text{CQ}(\text{glvn}_3, \text{glvn}_2) = \text{glvn}_2$