# System Library Reference

August 7, 2016

# Contents


**Description** xvii

**Copyrights** xviii

**Namespaces** xxi

**1 Global Namespace** 1
- 1.1 Concepts . . . 2
  - 1.1.1 Common<T, U> Concept . . . 3
  - 1.1.2 Convertible<T, U> Concept . . . 4
  - 1.1.3 Derived<T, U> Concept . . . 5
  - 1.1.4 ExplicitlyConvertible<T, U> Concept . . . 6
  - 1.1.5 NonReferenceType<T> Concept . . . 7
  - 1.1.6 Same<T, U> Concept . . . 8

**2 System Namespace** 9
- 2.2 Classes . . . 12
  - 2.2.1 BackInsertIterator<C> Class . . . 16
    - 2.2.1.1 Remarks . . . 16
    - 2.2.1.2 Example . . . 16
    - 2.2.1.3 Type Definitions . . . 17
    - 2.2.1.4 Member Functions . . . 17
  - 2.2.2 BackInsertProxy<C> Class . . . 27
    - 2.2.2.1 Member Functions . . . 27
  - 2.2.3 BinaryFun<Argument1, Argument2, Result> Class . . . 30
    - 2.2.3.1 Remarks . . . 30
    - 2.2.3.2 Type Definitions . . . 31
    - 2.2.3.3 Member Functions . . . 31
  - 2.2.4 BinaryPred<Argument1, Argument2> Class . . . 37
    - 2.2.4.1 Remarks . . . 37
    - 2.2.4.2 Member Functions . . . 37
  - 2.2.5 BitAnd<T> Class . . . 43
    - 2.2.5.1 Member Functions . . . 43
  - 2.2.6 BitNot<T> Class . . . 50
    - 2.2.6.1 Member Functions . . . 50
  - 2.2.7 BitOr<T> Class . . . 57
    - 2.2.7.1 Member Functions . . . 57
  - 2.2.8 BitXor<T> Class . . . 64
    - 2.2.8.1 Member Functions . . . 64
  - 2.2.9 Console Class . . . 71
    - 2.2.9.1 Remarks . . . 71
    - 2.2.9.2 Example . . . 71

2.2.9.3 Member Functions . . . 71
2.2.10 ConversionException Class . . . 120
2.2.10.1 Remarks . . . 120
2.2.10.2 Example . . . 120
2.2.10.3 Member Functions . . . 120
2.2.11 Counter<T> Class . . . 127
2.2.11.1 Remarks . . . 127
2.2.11.2 Member Functions . . . 127
2.2.12 CounterBase Class . . . 131
2.2.12.1 Member Functions . . . 131
2.2.13 Date Class . . . 141
2.2.13.1 Remarks . . . 141
2.2.13.2 Member Functions . . . 141
2.2.13.3 Nonmember Functions . . . 150
2.2.14 Divides<T> Class . . . 153
2.2.14.1 Example . . . 153
2.2.14.2 Member Functions . . . 153
2.2.15 Duration Class . . . 161
2.2.15.1 Member Functions . . . 161
2.2.15.2 Nonmember Functions . . . 181
2.2.16 EndLine Class . . . 190
2.2.16.1 Remarks . . . 190
2.2.16.2 Member Functions . . . 190
2.2.17 EqualTo2<T, U> Class . . . 194
2.2.17.1 Remarks . . . 194
2.2.17.2 Member Functions . . . 194
2.2.18 EqualTo<T> Class . . . 201
2.2.18.1 Member Functions . . . 201
2.2.19 Exception Class . . . 208
2.2.19.1 Remarks . . . 208
2.2.19.2 Example . . . 208
2.2.19.3 Member Functions . . . 208
2.2.20 ExceptionPtr Class . . . 226
2.2.20.1 Remarks . . . 226
2.2.20.2 Example . . . 226
2.2.20.3 Member Functions . . . 227
2.2.21 FrontInsertIterator<C> Class . . . 237
2.2.21.1 Remarks . . . 237
2.2.21.2 Example . . . 237
2.2.21.3 Type Definitions . . . 238
2.2.21.4 Member Functions . . . 238
2.2.22 FrontInsertProxy<C> Class . . . 248
2.2.22.1 Member Functions . . . 248
2.2.23 Greater2<T, U> Class . . . 251
2.2.23.1 Remarks . . . 251
2.2.23.2 Member Functions . . . 251
2.2.24 Greater<T> Class . . . 258
2.2.24.1 Example . . . 258
2.2.24.2 Example . . . 258
2.2.24.3 Member Functions . . . 259
2.2.25 GreaterOrEqualTo2<T, U> Class . . . 267
2.2.25.1 Remarks . . . 267
2.2.25.2 Member Functions . . . 267
2.2.26 GreaterOrEqualTo<T> Class . . . 274

2.2.26.1 Member Functions . . . 274
2.2.27 Handle Class . . . 281
2.2.27.1 Member Functions . . . 281
2.2.28 Identity<T> Class . . . 287
2.2.28.1 Member Functions . . . 287
2.2.29 InsertIterator<C> Class . . . 294
2.2.29.1 Remarks . . . 294
2.2.29.2 Example . . . 294
2.2.29.3 Type Definitions . . . 295
2.2.29.4 Member Functions . . . 295
2.2.30 InsertProxy<C> Class . . . 304
2.2.30.1 Member Functions . . . 304
2.2.31 Less2<T, U> Class . . . 307
2.2.31.1 Remarks . . . 307
2.2.31.2 Member Functions . . . 307
2.2.32 Less<T> Class . . . 314
2.2.32.1 Example . . . 314
2.2.32.2 Example . . . 314
2.2.32.3 Member Functions . . . 315
2.2.33 LessOrEqualTo2<T, U> Class . . . 323
2.2.33.1 Remarks . . . 323
2.2.33.2 Member Functions . . . 323
2.2.34 LessOrEqualTo<T> Class . . . 330
2.2.34.1 Member Functions . . . 330
2.2.35 LogicalAnd<T> Class . . . 337
2.2.35.1 Member Functions . . . 337
2.2.36 LogicalNot<T> Class . . . 344
2.2.36.1 Member Functions . . . 344
2.2.37 LogicalOr<T> Class . . . 351
2.2.37.1 Member Functions . . . 351
2.2.38 MT Class . . . 358
2.2.38.1 Remarks . . . 358
2.2.38.2 Member Functions . . . 358
2.2.39 Minus<T> Class . . . 362
2.2.39.1 Member Functions . . . 362
2.2.40 Multiplies<T> Class . . . 369
2.2.40.1 Example . . . 369
2.2.40.2 Member Functions . . . 369
2.2.41 Negate<T> Class . . . 377
2.2.41.1 Member Functions . . . 377
2.2.42 NotEqualTo2<T, U> Class . . . 384
2.2.42.1 Remarks . . . 384
2.2.42.2 Member Functions . . . 384
2.2.43 NotEqualTo<T> Class . . . 391
2.2.43.1 Member Functions . . . 391
2.2.44 Pair<T, U> Class . . . 398
2.2.44.1 Remarks . . . 398
2.2.44.2 Member Functions . . . 398
2.2.44.3 Nonmember Functions . . . 404
2.2.45 Plus<T> Class . . . 407
2.2.45.1 Example . . . 407
2.2.45.2 Member Functions . . . 407
2.2.46 RandomAccessIter<T, R, P> Class . . . 415
2.2.46.1 Remarks . . . 415

2.2.46.2 Type Definitions . . . 416
2.2.46.3 Member Functions . . . 416
2.2.46.4 Nonmember Functions . . . 428
2.2.47 Rel<Argument> Class . . . 435
2.2.47.1 Remarks . . . 435
2.2.47.2 Type Definitions . . . 436
2.2.47.3 Member Functions . . . 436
2.2.48 Remainder<T> Class . . . 442
2.2.48.1 Member Functions . . . 442
2.2.49 SelectFirst<T, U> Class . . . 449
2.2.49.1 Member Functions . . . 449
2.2.50 SelectSecond<T, U> Class . . . 456
2.2.50.1 Member Functions . . . 456
2.2.51 ShareableFromThis<T> Class . . . 463
2.2.51.1 Remarks . . . 463
2.2.51.2 Example . . . 463
2.2.51.3 Member Functions . . . 463
2.2.52 SharedCount<T> Class . . . 472
2.2.52.1 Member Functions . . . 472
2.2.52.2 Nonmember Functions . . . 483
2.2.53 SharedPtr<T> Class . . . 486
2.2.53.1 Example . . . 486
2.2.53.2 Member Functions . . . 487
2.2.53.3 Nonmember Functions . . . 504
2.2.54 ShiftLeft<T> Class . . . 507
2.2.54.1 Member Functions . . . 507
2.2.55 ShiftRight<T> Class . . . 514
2.2.55.1 Member Functions . . . 514
2.2.56 String<CharT> Class . . . 521
2.2.56.1 Type Definitions . . . 522
2.2.56.2 Member Functions . . . 522
2.2.56.3 Nonmember Functions . . . 571
2.2.57 TimeError Class . . . 575
2.2.57.1 Member Functions . . . 575
2.2.58 TimePoint Class . . . 583
2.2.58.1 Remarks . . . 583
2.2.58.2 Member Functions . . . 583
2.2.58.3 Nonmember Functions . . . 590
2.2.59 TracedFun Class . . . 596
2.2.59.1 Member Functions . . . 596
2.2.60 Tracer Class . . . 600
2.2.60.1 Member Functions . . . 600
2.2.61 UnaryFun<Argument, Result> Class . . . 604
2.2.61.1 Remarks . . . 604
2.2.61.2 Type Definitions . . . 605
2.2.61.3 Member Functions . . . 605
2.2.62 UnaryPred<Argument> Class . . . 611
2.2.62.1 Remarks . . . 611
2.2.62.2 Member Functions . . . 611
2.2.63 UniquePtr<T> Class . . . 617
2.2.63.1 Remarks . . . 617
2.2.63.2 Example . . . 617
2.2.63.3 Member Functions . . . 617
2.2.63.4 Nonmember Functions . . . 632

2.2.64 WeakCount<T> Class . . . 635
2.2.64.1 Member Functions . . . 635
2.2.64.2 Nonmember Functions . . . 644
2.2.65 WeakPtr<T> Class . . . 647
2.2.65.1 Remarks . . . 647
2.2.65.2 Example . . . 647
2.2.65.3 Member Functions . . . 649
2.2.66 uhuge Class . . . 667
2.2.66.1 Member Functions . . . 667
2.2.66.2 Nonmember Functions . . . 676
2.3 Type Definitions . . . 694
2.4 Functions . . . 695
2.4.67 Abs(const T&) Function . . . 706
2.4.67.1 Example . . . 706
2.4.68 Accumulate(I, I, T, Op) Function . . . 707
2.4.68.1 Example . . . 707
2.4.69 BackInserter(C&) Function . . . 709
2.4.69.1 Example . . . 709
2.4.70 CaptureCurrentException() Function . . . 710
2.4.70.1 Example . . . 710
2.4.71 Copy(I, I, O) Function . . . 712
2.4.71.1 Example . . . 712
2.4.72 CopyBackward(I, I, O) Function . . . 714
2.4.72.1 Example . . . 714
2.4.73 Count(I, I, P) Function . . . 716
2.4.73.1 Example . . . 716
2.4.74 Count(I, I, const T&) Function . . . 718
2.4.74.1 Example . . . 718
2.4.75 CurrentDate() Function . . . 720
2.4.76 Distance(I, I) Function . . . 721
2.4.76.1 Example . . . 721
2.4.77 Distance(I, I) Function . . . 722
2.4.78 EnableSharedFromThis(System.ShareableFromThis<T>*, U*, const System.SharedCount<U>&) Function . . . 723
2.4.79 EnableSharedFromThis(void*, void*, const System.SharedCount<T>&) Function . . . 724
2.4.80 Equal(I1, I1, I2, I2) Function . . . 725
2.4.80.1 Example . . . 725
2.4.81 Equal(I1, I1, I2, I2, R) Function . . . 727
2.4.81.1 Example . . . 727
2.4.82 EqualRange(I, I, const T&) Function . . . 729
2.4.82.1 Example . . . 729
2.4.82.2 Example . . . 730
2.4.83 EqualRange(I, I, const T&, R) Function . . . 731
2.4.83.1 Example . . . 731
2.4.84 Execute(const System.String<char>&) Function . . . 733
2.4.85 Execute(const System.String<char>&, bool) Function . . . 734
2.4.86 Execute(const System.String<char>&, const System.Collections.List<System.Pair<int, System.String<char>>>&) Function . . . 735
2.4.87 Execute(const System.String<char>&, const System.Collections.List<System.Pair<int, System.String<char>>>&, bool) Function . . . 736
2.4.88 Execute(const System.String<char>&, int, const System.String<char>&) Function . . . 737

2.4.89 Execute(const System.String<char>&, int, const System.String<char>&, bool) Function . . . 738
2.4.90 Factorial(U) Function . . . 739
2.4.90.1 Example . . . 739
2.4.91 Fill(I, I, const T&) Function . . . 740
2.4.92 Find(I, I, P) Function . . . 741
2.4.92.1 Example . . . 741
2.4.93 Find(I, I, const T&) Function . . . 743
2.4.93.1 Example . . . 743
2.4.94 ForEach(I, I, F) Function . . . 745
2.4.94.1 Example . . . 745
2.4.95 FrontInserter(C&) Function . . . 747
2.4.95.1 Example . . . 747
2.4.96 Gcd(T, T) Function . . . 748
2.4.96.1 Example . . . 748
2.4.97 HexChar(byte) Function . . . 749
2.4.98 IdentityElement(System.Multiplies<T>) Function . . . 750
2.4.99 IdentityElement(System.Plus<T>) Function . . . 751
2.4.100 Inserter(C&, I) Function . . . 752
2.4.100.1 Example . . . 752
2.4.101 InsertionSort(I, I) Function . . . 753
2.4.101.1 Example . . . 753
2.4.102 InsertionSort(I, I, R) Function . . . 754
2.4.102.1 Example . . . 754
2.4.103 IsAlpha(char) Function . . . 755
2.4.104 IsAlphanumeric(char) Function . . . 756
2.4.105 IsControl(char) Function . . . 757
2.4.106 IsDigit(char) Function . . . 758
2.4.107 IsGraphic(char) Function . . . 759
2.4.108 IsHexDigit(char) Function . . . 760
2.4.109 IsLower(char) Function . . . 761
2.4.110 IsPrintable(char) Function . . . 762
2.4.111 IsPunctuation(char) Function . . . 763
2.4.112 IsSpace(char) Function . . . 764
2.4.113 IsUpper(char) Function . . . 765
2.4.114 LastComponentsEqual(const System.String<CharT>&, const System.String<CharT>&, CharT) Function . . . 766
2.4.115 LexicographicalCompare(I1, I1, I2, I2) Function . . . 767
2.4.115.1 Example . . . 767
2.4.116 LexicographicalCompare(I1, I1, I2, I2, R) Function . . . 769
2.4.116.1 Example . . . 769
2.4.117 LowerBound(I, I, const T&) Function . . . 771
2.4.117.1 Example . . . 771
2.4.118 LowerBound(I, I, const T&, R) Function . . . 773
2.4.119 MakePair(const T&, const U&) Function . . . 774
2.4.120 Max(const T&, const T&) Function . . . 775
2.4.121 MaxElement(I, I) Function . . . 776
2.4.121.1 Example . . . 776
2.4.122 MaxElement(I, I, R) Function . . . 778
2.4.123 MaxValue() Function . . . 779
2.4.124 MaxValue(byte) Function . . . 780
2.4.125 MaxValue(int) Function . . . 781
2.4.126 MaxValue(long) Function . . . 782
2.4.127 MaxValue(sbyte) Function . . . 783

2.4.128 MaxValue(short) Function . . . 784
2.4.129 MaxValue(uint) Function . . . 785
2.4.130 MaxValue(ulong) Function . . . 786
2.4.131 MaxValue(ushort) Function . . . 787
2.4.132 Median(const T&, const T&, const T&) Function . . . 788
2.4.133 Median(const T&, const T&, const T&, R) Function . . . 789
2.4.134 Min(const T&, const T&) Function . . . 790
2.4.135 MinElement(I, I) Function . . . 791
2.4.135.1 Example . . . 791
2.4.136 MinElement(I, I, R) Function . . . 793
2.4.137 MinValue() Function . . . 794
2.4.138 MinValue(byte) Function . . . 795
2.4.139 MinValue(int) Function . . . 796
2.4.140 MinValue(long) Function . . . 797
2.4.141 MinValue(sbyte) Function . . . 798
2.4.142 MinValue(short) Function . . . 799
2.4.143 MinValue(uint) Function . . . 800
2.4.144 MinValue(ulong) Function . . . 801
2.4.145 MinValue(ushort) Function . . . 802
2.4.146 Move(I, I, O) Function . . . 803
2.4.147 MoveBackward(I, I, O) Function . . . 804
2.4.148 Next(I, int) Function . . . 805
2.4.149 Next(I, int) Function . . . 806
2.4.150 NextPermutation(I, I) Function . . . 807
2.4.150.1 Example . . . 807
2.4.151 NextPermutation(I, I, R) Function . . . 809
2.4.152 Now() Function . . . 810
2.4.153 ParseBool(const System.String<char>&) Function . . . 811
2.4.154 ParseBool(const System.String<char>&, bool&) Function . . . 812
2.4.155 ParseDate(const System.String<char>&) Function . . . 813
2.4.156 ParseDouble(const System.String<char>&) Function . . . 814
2.4.157 ParseDouble(const System.String<char>&, double&) Function . . . 815
2.4.158 ParseHex(const System.String<char>&) Function . . . 816
2.4.159 ParseHex(const System.String<char>&, System.uhuge&) Function . . . 817
2.4.160 ParseHex(const System.String<char>&, ulong&) Function . . . 818
2.4.161 ParseHexUHuge(const System.String<char>&) Function . . . 819
2.4.162 ParseInt(const System.String<char>&) Function . . . 820
2.4.163 ParseInt(const System.String<char>&, int&) Function . . . 821
2.4.164 ParseLong(const System.String<char>&) Function . . . 822
2.4.165 ParseLong(const System.String<char>&, long&) Function . . . 823
2.4.166 ParseUHuge(const System.String<char>&) Function . . . 824
2.4.167 ParseUHuge(const System.String<char>&, System.uhuge&) Function . . . 825
2.4.168 ParseUInt(const System.String<char>&) Function . . . 826
2.4.169 ParseUInt(const System.String<char>&, uint&) Function . . . 827
2.4.170 ParseULong(const System.String<char>&) Function . . . 828
2.4.171 ParseULong(const System.String<char>&, ulong&) Function . . . 829
2.4.172 PrevPermutation(I, I) Function . . . 830
2.4.173 PrevPermutation(I, I, R) Function . . . 831
2.4.174 PtrCast(const System.SharedPtr<T>&) Function . . . 832
2.4.174.1 Example . . . 832
2.4.175 Rand() Function . . . 833
2.4.176 RandomNumber(uint) Function . . . 834
2.4.177 RandomShuffle(I, I) Function . . . 835
2.4.178 Remove(I, I, P) Function . . . 836

2.4.179 Remove(I, I, const T&) Function . . . 837
2.4.180 RemoveCopy(I, I, O, P) Function . . . 838
2.4.181 RemoveCopy(I, I, O, const T&) Function . . . 839
2.4.182 RethrowException(System.ExceptionPtr&) Function . . . 840
2.4.182.1 Example . . . 840
2.4.183 Reverse(I, I) Function . . . 842
2.4.183.1 Example . . . 842
2.4.184 Reverse(I, I) Function . . . 843
2.4.185 ReverseUntil(I, I, I) Function . . . 844
2.4.186 Rotate(I, I, I) Function . . . 845
2.4.187 Rvalue(T&&) Function . . . 846
2.4.187.1 Example . . . 846
2.4.188 Select_0_2(const T&, const T&, R) Function . . . 848
2.4.189 Select_0_3(const T&, const T&, const T&, R) Function . . . 849
2.4.190 Select_1_2(const T&, const T&, R) Function . . . 850
2.4.191 Select_1_3(const T&, const T&, const T&, R) Function . . . 851
2.4.192 Select_1_3_ab(const T&, const T&, const T&, R) Function . . . 852
2.4.193 Select_2_3(const T&, const T&, const T&, R) Function . . . 853
2.4.194 Sort(C&) Function . . . 854
2.4.194.1 Example . . . 854
2.4.195 Sort(C&) Function . . . 855
2.4.196 Sort(C&, R) Function . . . 856
2.4.196.1 Example . . . 856
2.4.197 Sort(C&, R) Function . . . 857
2.4.198 Sort(I, I) Function . . . 858
2.4.199 Sort(I, I, R) Function . . . 859
2.4.200 Swap(T&, T&) Function . . . 860
2.4.201 ThrowConversionException(const System.String<char>&) Function . . . 861
2.4.202 ToHexString(System.uhuge) Function . . . 862
2.4.203 ToHexString(U) Function . . . 863
2.4.204 ToHexString(byte) Function . . . 864
2.4.205 ToHexString(uint) Function . . . 865
2.4.206 ToHexString(ulong) Function . . . 866
2.4.207 ToHexString(ushort) Function . . . 867
2.4.208 ToLower(const System.String<char>&) Function . . . 868
2.4.209 ToLower(const System.String<uchar>&) Function . . . 869
2.4.210 ToLower(const System.String<wchar>&) Function . . . 870
2.4.211 ToString(I) Function . . . 871
2.4.212 ToString(System.Date) Function . . . 872
2.4.213 ToString(System.uhuge) Function . . . 873
2.4.214 ToString(U) Function . . . 874
2.4.215 ToString(bool) Function . . . 875
2.4.216 ToString(byte) Function . . . 876
2.4.217 ToString(char) Function . . . 877
2.4.218 ToString(double) Function . . . 878
2.4.219 ToString(double, int) Function . . . 879
2.4.220 ToString(double, int, int) Function . . . 880
2.4.221 ToString(int) Function . . . 881
2.4.222 ToString(long) Function . . . 882
2.4.223 ToString(sbyte) Function . . . 883
2.4.224 ToString(short) Function . . . 884
2.4.225 ToString(uchar) Function . . . 885
2.4.226 ToString(uint) Function . . . 886
2.4.227 ToString(ulong) Function . . . 887

2.4.228 ToString(ushort) Function . . . 888
2.4.229 ToString(wchar) Function . . . 889
2.4.230 ToUpper(const System.String<char>&) Function . . . 890
2.4.231 ToUpper(const System.String<uchar>&) Function . . . 891
2.4.232 ToUpper(const System.String<wchar>&) Function . . . 892
2.4.233 Transform(I, I, O, F) Function . . . 893
2.4.233.1 Example . . . 893
2.4.234 Transform(I1, I1, I2, O, F) Function . . . 895
2.4.234.1 Example . . . 895
2.4.235 UpperBound(I, I, const T&) Function . . . 897
2.4.235.1 Example . . . 897
2.4.236 UpperBound(I, I, const T&, R) Function . . . 899
2.4.237 endl() Function . . . 900
2.5 Enumerations . . . 901
2.5.237.1 CharClass Enumeration . . . 902
2.6 Constants . . . 903

**3 System.Collections Namespace 904**
3.7 Classes . . . 905
3.7.1 BitSet Class . . . 907
3.7.1.1 Example . . . 907
3.7.1.2 Member Functions . . . 908
3.7.2 Bucket<T> Class . . . 935
3.7.2.1 Type Definitions . . . 936
3.7.2.2 Member Functions . . . 936
3.7.3 ForwardList<T> Class . . . 946
3.7.3.1 Remarks . . . 946
3.7.3.2 Type Definitions . . . 947
3.7.3.3 Member Functions . . . 947
3.7.3.4 Nonmember Functions . . . 969
3.7.4 ForwardListNode<T> Class . . . 972
3.7.4.1 Member Functions . . . 972
3.7.5 ForwardListNodeIterator<T, R, P> Class . . . 983
3.7.5.1 Type Definitions . . . 984
3.7.5.2 Member Functions . . . 984
3.7.5.3 Nonmember Functions . . . 994
3.7.6 HashMap<K, T, H, C> Class . . . 996
3.7.6.1 Type Definitions . . . 997
3.7.6.2 Member Functions . . . 997
3.7.6.3 Nonmember Functions . . . 1020
3.7.7 HashSet<T, H, C> Class . . . 1022
3.7.7.1 Type Definitions . . . 1023
3.7.7.2 Member Functions . . . 1023
3.7.7.3 Nonmember Functions . . . 1044
3.7.8 Hasher<T> Class . . . 1046
3.7.8.1 Member Functions . . . 1046
3.7.9 Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare> Class 1053
3.7.9.1 Type Definitions . . . 1054
3.7.9.2 Member Functions . . . 1054
3.7.10 HashtableBase<T> Class . . . 1081
3.7.10.1 Type Definitions . . . 1082
3.7.10.2 Member Functions . . . 1082
3.7.11 HashtableIterator<T, R, P> Class . . . 1083
3.7.11.1 Type Definitions . . . 1084

3.7.11.2 Member Functions . . . 1084
3.7.11.3 Nonmember Functions . . . 1093
3.7.12 LinkedList<T> Class . . . 1095
3.7.12.1 Type Definitions . . . 1096
3.7.12.2 Member Functions . . . 1096
3.7.12.3 Nonmember Functions . . . 1121
3.7.13 LinkedListBase Class . . . 1124
3.7.13.1 Member Functions . . . 1124
3.7.14 LinkedListNode<T> Class . . . 1125
3.7.14.1 Type Definitions . . . 1126
3.7.14.2 Member Functions . . . 1126
3.7.15 LinkedListNodeBase Class . . . 1135
3.7.15.1 Member Functions . . . 1135
3.7.16 LinkedListNodeIterator<T, R, P> Class . . . 1137
3.7.16.1 Type Definitions . . . 1138
3.7.16.2 Member Functions . . . 1138
3.7.16.3 Nonmember Functions . . . 1149
3.7.17 List<T> Class . . . 1151
3.7.17.1 Example . . . 1151
3.7.17.2 Type Definitions . . . 1152
3.7.17.3 Member Functions . . . 1152
3.7.17.4 Nonmember Functions . . . 1188
3.7.18 Map<Key, Value, KeyCompare> Class . . . 1191
3.7.18.1 Example . . . 1191
3.7.18.2 Type Definitions . . . 1193
3.7.18.3 Member Functions . . . 1193
3.7.18.4 Nonmember Functions . . . 1215
3.7.19 Queue<T> Class . . . 1220
3.7.19.1 Example . . . 1220
3.7.19.2 Type Definitions . . . 1222
3.7.19.3 Member Functions . . . 1222
3.7.20 RedBlackTree<KeyType, ValueType, KeyOfValue, Compare> Class . . . 1235
3.7.20.1 Type Definitions . . . 1236
3.7.20.2 Member Functions . . . 1236
3.7.21 RedBlackTreeNodeIterator<T, R, P> Class . . . 1258
3.7.21.1 Type Definitions . . . 1259
3.7.21.2 Member Functions . . . 1259
3.7.21.3 Nonmember Functions . . . 1270
3.7.22 Set<T, C> Class . . . 1272
3.7.22.1 Example . . . 1272
3.7.22.2 Type Definitions . . . 1274
3.7.22.3 Member Functions . . . 1274
3.7.22.4 Nonmember Functions . . . 1295
3.7.23 Stack<T> Class . . . 1300
3.7.23.1 Example . . . 1300
3.7.23.2 Type Definitions . . . 1301
3.7.23.3 Member Functions . . . 1301
3.8 Functions . . . 1315
3.8.24 ConstructiveCopy(ValueType*, ValueType*, int) Function . . . 1316
3.8.25 ConstructiveMove(ValueType*, ValueType*, int) Function . . . 1317
3.8.26 Destroy(ValueType*, int) Function . . . 1318
3.8.27 GetHashCode(char) Function . . . 1319
3.8.28 GetHashCode(long) Function . . . 1320
3.8.29 GetHashCode(uchar) Function . . . 1321

3.8.30 GetHashCode(ulong) Function . . . 1322
3.8.31 GetHashCode(void*) Function . . . 1323
3.8.32 GetHashCode(wchar) Function . . . 1324

**4 System.Concepts Namespace 1325**
4.9 Concepts . . . 1328
4.9.1 AdditiveGroup<T> Concept . . . 1333
4.9.2 AdditiveMonoid<T> Concept . . . 1334
4.9.3 AdditiveSemigroup<T> Concept . . . 1335
4.9.4 BackInsertionSequence<T> Concept . . . 1336
4.9.5 BidirectionalContainer<T> Concept . . . 1337
4.9.6 BidirectionalIterator<T> Concept . . . 1338
4.9.7 BinaryFunction<T> Concept . . . 1339
4.9.8 BinaryOperation<T> Concept . . . 1340
4.9.9 BinaryPredicate<T> Concept . . . 1341
4.9.10 CommutativeSemiring<T> Concept . . . 1342
4.9.11 Container<T> Concept . . . 1343
4.9.12 ConversionFromByte<T> Concept . . . 1344
4.9.13 ConversionFromSByte<T> Concept . . . 1345
4.9.14 CopyAssignable<T, U> Concept . . . 1346
4.9.14.1 Example . . . 1346
4.9.15 CopyAssignable<T> Concept . . . 1347
4.9.15.1 Example . . . 1347
4.9.16 CopyConstructible<T> Concept . . . 1349
4.9.16.1 Example . . . 1349
4.9.17 Copyable<T> Concept . . . 1351
4.9.18 DefaultConstructible<T> Concept . . . 1352
4.9.18.1 Example . . . 1352
4.9.19 Destructible<T> Concept . . . 1354
4.9.19.1 Example . . . 1354
4.9.20 EqualityComparable<T, U> Concept . . . 1356
4.9.21 EqualityComparable<T> Concept . . . 1357
4.9.22 EuclideanSemiring<T> Concept . . . 1358
4.9.23 ForwardContainer<T> Concept . . . 1359
4.9.24 ForwardIterator<T> Concept . . . 1360
4.9.25 FrontInsertionSequence<T> Concept . . . 1361
4.9.26 HashFunction<T, Key> Concept . . . 1362
4.9.27 InputIterator<T> Concept . . . 1363
4.9.28 InsertionSequence<T> Concept . . . 1364
4.9.29 Integer<I> Concept . . . 1365
4.9.30 KeySelectionFunction<T, Key, Value> Concept . . . 1366
4.9.31 LessThanComparable<T, U> Concept . . . 1367
4.9.32 LessThanComparable<T> Concept . . . 1368
4.9.33 Movable<T> Concept . . . 1369
4.9.34 MoveAssignable<T> Concept . . . 1370
4.9.35 MoveConstructible<T> Concept . . . 1371
4.9.36 MultiplicativeGroup<T> Concept . . . 1372
4.9.37 MultiplicativeMonoid<T> Concept . . . 1373
4.9.38 MultiplicativeSemigroup<T> Concept . . . 1374
4.9.39 OrderedAdditiveGroup<T> Concept . . . 1375
4.9.40 OrderedAdditiveMonoid<T> Concept . . . 1376
4.9.41 OrderedAdditiveSemigroup<T> Concept . . . 1377
4.9.42 OrderedMultiplicativeSemigroup<T> Concept . . . 1378
4.9.43 OutputIterator<T> Concept . . . 1379

4.9.44 RandomAccessContainer<T> Concept . . . 1380
4.9.45 RandomAccessIterator<T> Concept . . . 1381
4.9.46 Regular<T> Concept . . . 1382
4.9.46.1 Example . . . 1382
4.9.47 Relation<T, U, V> Concept . . . 1385
4.9.48 Relation<T> Concept . . . 1386
4.9.49 Semiregular<T> Concept . . . 1387
4.9.49.1 Example . . . 1387
4.9.50 Semiring<T> Concept . . . 1389
4.9.51 SignedInteger<I> Concept . . . 1390
4.9.52 TotallyOrdered<T, U> Concept . . . 1391
4.9.53 TotallyOrdered<T> Concept . . . 1392
4.9.53.1 Example . . . 1392
4.9.54 TrivialIterator<T> Concept . . . 1396
4.9.55 UnaryFunction<T> Concept . . . 1397
4.9.56 UnaryOperation<T> Concept . . . 1398
4.9.57 UnaryPredicate<T> Concept . . . 1399
4.9.58 UnsignedInteger<U> Concept . . . 1400

**5 System.IO Namespace 1401**
5.10 Classes . . . 1402
5.10.1 BinaryFileStream Class . . . 1403
5.10.1.1 Member Functions . . . 1403
5.10.2 ByteStream Class . . . 1442
5.10.2.1 Member Functions . . . 1442
5.10.3 CloseFileException Class . . . 1452
5.10.3.1 Member Functions . . . 1452
5.10.4 FileByteStream Class . . . 1459
5.10.4.1 Member Functions . . . 1459
5.10.5 FileMapping Class . . . 1467
5.10.5.1 Member Functions . . . 1467
5.10.6 FileMappingFailure Class . . . 1474
5.10.6.1 Member Functions . . . 1474
5.10.7 IOBuffer Class . . . 1480
5.10.7.1 Member Functions . . . 1480
5.10.8 IOException Class . . . 1487
5.10.8.1 Member Functions . . . 1487
5.10.9 InputFileStream Class . . . 1494
5.10.9.1 Member Functions . . . 1494
5.10.10 InputStream Class . . . 1507
5.10.10.1 Member Functions . . . 1507
5.10.11 InputStringStream Class . . . 1515
5.10.11.1 Member Functions . . . 1515
5.10.12 InvalidPathException Class . . . 1526
5.10.12.1 Member Functions . . . 1526
5.10.13 MemoryByteStream Class . . . 1534
5.10.13.1 Member Functions . . . 1534
5.10.14 OpenFileException Class . . . 1548
5.10.14.1 Member Functions . . . 1548
5.10.15 OutputFileStream Class . . . 1556
5.10.15.1 Member Functions . . . 1556
5.10.16 OutputStream Class . . . 1613
5.10.16.1 Member Functions . . . 1613
5.10.16.2 Nonmember Functions . . . 1656

5.10.17 OutputStringStream Class . . . 1679
5.10.17.1 Member Functions . . . 1679
5.10.18 Path Class . . . 1726
5.10.18.1 Member Functions . . . 1726
5.11 Functions . . . 1738
5.11.19 CreateDirectories(const System.String<char>&) Function . . . 1739
5.11.20 DirectoryExists(const System.String<char>&) Function . . . 1740
5.11.21 FileContentsEqual(const System.String<char>&, const System.String<char>&) Function . . . 1741
5.11.22 FileExists(const System.String<char>&) Function . . . 1742
5.11.23 GetCurrentWorkingDirectory() Function . . . 1743
5.11.24 GetFullPath(const System.String<char>&) Function . . . 1744
5.11.25 GetPathToExecutable() Function . . . 1745
5.11.26 LastWriteTimeLess(const System.String<char>&, const System.String<char>&) Function . . . 1746
5.11.27 PathExists(const System.String<char>&) Function . . . 1747
5.11.28 ReadFile(const System.String<char>&) Function . . . 1748
5.11.29 RemoveFile(const System.String<char>&) Function . . . 1749
5.12 Enumerations . . . 1750
5.12.29.1 FileMode Enumeration . . . 1751
5.12.29.2 OpenMode Enumeration . . . 1751

**6 System.Meta Namespace 1752**
6.13 Functions . . . 1753
6.13.1 IsBasicType<T0>() Function . . . 1757
6.13.2 IsBoolType<T0>() Function . . . 1758
6.13.3 IsByteType<T0>() Function . . . 1759
6.13.4 IsCharType<T0>() Function . . . 1760
6.13.5 IsCharacterType<T0>() Function . . . 1761
6.13.6 IsClassDelegateType<T0>() Function . . . 1762
6.13.7 IsClassType<T0>() Function . . . 1763
6.13.8 IsConstType<T0>() Function . . . 1764
6.13.9 IsDelegateType<T0>() Function . . . 1765
6.13.10 IsDoubleType<T0>() Function . . . 1766
6.13.11 IsEnumType<T0>() Function . . . 1767
6.13.12 IsFloatType<T0>() Function . . . 1768
6.13.13 IsFloatingPointType<T0>() Function . . . 1769
6.13.14 IsIntType<T0>() Function . . . 1770
6.13.15 IsIntegerType<T0>() Function . . . 1771
6.13.16 IsLongType<T0>() Function . . . 1772
6.13.17 IsLvalueReferenceType<T0>() Function . . . 1773
6.13.18 IsNonBasicType<T0>() Function . . . 1774
6.13.19 IsNonBoolType<T0>() Function . . . 1775
6.13.20 IsNonByteType<T0>() Function . . . 1776
6.13.21 IsNonCharType<T0>() Function . . . 1777
6.13.22 IsNonCharacterType<T0>() Function . . . 1778
6.13.23 IsNonClassDelegateType<T0>() Function . . . 1779
6.13.24 IsNonClassType<T0>() Function . . . 1780
6.13.25 IsNonConstType<T0>() Function . . . 1781
6.13.26 IsNonDelegateType<T0>() Function . . . 1782
6.13.27 IsNonDoubleType<T0>() Function . . . 1783
6.13.28 IsNonEnumType<T0>() Function . . . 1784
6.13.29 IsNonFloatType<T0>() Function . . . 1785
6.13.30 IsNonFloatingPointType<T0>() Function . . . 1786

6.13.31 IsNonIntType<T0>() Function . . . 1787
6.13.32 IsNonIntegerType<T0>() Function . . . 1788
6.13.33 IsNonLongType<T0>() Function . . . 1789
6.13.34 IsNonLvalueReferenceType<T0>() Function . . . 1790
6.13.35 IsNonPointerType<T0>() Function . . . 1791
6.13.36 IsNonPolymorphic<T0>() Function . . . 1792
6.13.37 IsNonReferenceType<T0>() Function . . . 1793
6.13.38 IsNonRvalueReferenceType<T0>() Function . . . 1794
6.13.39 IsNonSByteType<T0>() Function . . . 1795
6.13.40 IsNonShortType<T0>() Function . . . 1796
6.13.41 IsNonSignedType<T0>() Function . . . 1797
6.13.42 IsNonUCharType<T0>() Function . . . 1798
6.13.43 IsNonUIntType<T0>() Function . . . 1799
6.13.44 IsNonULongType<T0>() Function . . . 1800
6.13.45 IsNonUShortType<T0>() Function . . . 1801
6.13.46 IsNonUnsignedType<T0>() Function . . . 1802
6.13.47 IsNonVoidType<T0>() Function . . . 1803
6.13.48 IsNonWCharType<T0>() Function . . . 1804
6.13.49 IsPointerType<T0>() Function . . . 1805
6.13.50 IsPolymorphic<T0>() Function . . . 1806
6.13.51 IsReferenceType<T0>() Function . . . 1807
6.13.52 IsRvalueReferenceType<T0>() Function . . . 1808
6.13.53 IsSByteType<T0>() Function . . . 1809
6.13.54 IsShortType<T0>() Function . . . 1810
6.13.55 IsSignedType<T0>() Function . . . 1811
6.13.56 IsUCharType<T0>() Function . . . 1812
6.13.57 IsUIntType<T0>() Function . . . 1813
6.13.58 IsULongType<T0>() Function . . . 1814
6.13.59 IsUShortType<T0>() Function . . . 1815
6.13.60 IsUnsignedType<T0>() Function . . . 1816
6.13.61 IsVoidType<T0>() Function . . . 1817
6.13.62 IsWCharType<T0>() Function . . . 1818

**7 System.Security Namespace 1819**
7.14 Classes . . . 1820
7.14.1 Sha1 Class . . . 1821
7.14.1.1 Member Functions . . . 1821
7.14.2 Sha256 Class . . . 1832
7.14.2.1 Member Functions . . . 1832
7.14.3 Sha512 Class . . . 1843
7.14.3.1 Member Functions . . . 1843
7.15 Functions . . . 1854
7.15.4 GetSha1FileDigest(const System.String<char>&) Function . . . 1855
7.15.5 GetSha1MessageDigest(const System.String<char>&) Function . . . 1856
7.15.6 GetSha256FileDigest(const System.String<char>&) Function . . . 1857
7.15.7 GetSha256MessageDigest(const System.String<char>&) Function . . . 1858
7.15.8 GetSha512FileDigest(const System.String<char>&) Function . . . 1859
7.15.9 GetSha512MessageDigest(const System.String<char>&) Function . . . 1860
7.15.10 LeftRotate(uint, uint) Function . . . 1861
7.15.11 RightRotate(uint, uint) Function . . . 1862
7.15.12 RightRotate(ulong, ulong) Function . . . 1863

**8 System.Text Namespace** **1864**
8.16 Classes . . . 1865
8.16.1 CodeFormatter Class . . . 1866
8.16.1.1 Member Functions . . . 1866
8.17 Functions . . . 1880
8.17.2 CharStr(char) Function . . . 1881
8.17.3 HexEscape(char) Function . . . 1882
8.17.4 HexEscape(uchar) Function . . . 1883
8.17.5 HexEscape(wchar) Function . . . 1884
8.17.6 MakeCharLiteral(char) Function . . . 1885
8.17.7 MakeStringLiteral(const System.String<char>&) Function . . . 1886
8.17.8 StringStr(const System.String<char>&) Function . . . 1887
8.17.9 Trim(const System.String<char>&) Function . . . 1888
8.17.10 TrimAll(const System.String<char>&) Function . . . 1889

**9 System.Threading Namespace** **1890**
9.18 Concepts . . . 1891
9.18.1 Lockable<M> Concept . . . 1892
9.19 Classes . . . 1893
9.19.2 ConditionVariable Class . . . 1894
9.19.2.1 Member Functions . . . 1894
9.19.3 LockGuard<M> Class . . . 1902
9.19.3.1 Member Functions . . . 1902
9.19.4 Mutex Class . . . 1907
9.19.4.1 Remarks . . . 1907
9.19.4.2 Member Functions . . . 1907
9.19.5 RecursiveMutex Class . . . 1914
9.19.5.1 Member Functions . . . 1914
9.19.6 Thread Class . . . 1917
9.19.6.1 Member Functions . . . 1917
9.19.6.2 Nonmember Functions . . . 1925
9.19.7 ThreadingException Class . . . 1927
9.19.7.1 Member Functions . . . 1927
9.20 Functions . . . 1935
9.20.8 HardwareConcurrency() Function . . . 1936
9.20.9 SleepFor(System.Duration) Function . . . 1937
9.20.10 SleepUntil(System.TimePoint) Function . . . 1938
9.20.11 ThreadStart(void*) Function . . . 1939
9.21 Delegates . . . 1940
9.21.12 ThreadFun Delegate . . . 1941
9.22 Constants . . . 1942

**10 System.Unicode Namespace** **1943**
10.23 Classes . . . 1944
10.24 Functions . . . 1945
10.24.1 GetCategory(uchar) Function . . . 1947
10.24.2 GetCategoryName(System.Unicode.Category) Function . . . 1948
10.24.3 GetCharacterName(uchar) Function . . . 1949
10.24.4 IsLetter(uchar) Function . . . 1950
10.24.5 IsLower(uchar) Function . . . 1951
10.24.6 IsMark(uchar) Function . . . 1952
10.24.7 IsNumber(uchar) Function . . . 1953
10.24.8 IsPunctuation(uchar) Function . . . 1954
10.24.9 IsSeparator(uchar) Function . . . 1955

10.24.10 IsSymbol(uchar) Function . . . . . 1956
10.24.11 IsUpper(uchar) Function . . . . . 1957
10.24.12 ToLower(uchar) Function . . . . . 1958
10.24.13 ToUpper(uchar) Function . . . . . 1959
10.24.14 ToUtf16(const System.String<char>&) Function . . . . . 1960
10.24.15 ToUtf16(const System.String<uchar>&) Function . . . . . 1961
10.24.16 ToUtf32(const System.String<char>&) Function . . . . . 1962
10.24.17 ToUtf32(const System.String<wchar>&) Function . . . . . 1963
10.24.18 ToUtf8(const System.String<uchar>&) Function . . . . . 1964
10.24.19 ToUtf8(const System.String<wchar>&) Function . . . . . 1965
10.25 Enumerations . . . . . 1966
10.25.19.1 Category Enumeration . . . . . 1967

# Description

The System library is the run-time library for Cmajor programs. The System library is built on top of the Support library that contains the run-time support for Cmajor language implementation (primarily support for exception handling). The Support library in turn is built on top of the Os library that provides interface to operating system services. The Os library is built on top of the C run-time library for the platform. In Windows this is the C run-time library of the **mingw_w64's gcc** compiler and in Linux this the C run-time library of the **GNU/Linux gcc** compiler. Figure 1 illustrates the layered architecture of Cmajor.

Figure 1: Libraries

| |
|---|
| system |
| support |
| os |
| C run-time |

Note: You may want to enable Previous View and Next View commands in the PDF viewer for comfortable browsing experience in this document.

# Copyrights


```
==========================================================================
Copyright (c) 2012-2016 Seppo Laakko
http://sourceforge.net/projects/cmajor/

Distributed under the GNU General Public License, version 3 (GPLv3).
(See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html)

==========================================================================
```

```
* Copyright (c) 1994
* Hewlett-Packard Company
*
* Permission to use, copy, modify, distribute and sell this software
* and its documentation for any purpose is hereby granted without fee,
* provided that the above copyright notice appear in all copies and
* that both that copyright notice and this permission notice appear
* in supporting documentation.  Hewlett-Packard Company makes no
* representations about the suitability of this software for any
* purpose.  It is provided "as is" without express or implied warranty.
*
*
* Copyright (c) 1996,1997
* Silicon Graphics Computer Systems, Inc.
*
* Permission to use, copy, modify, distribute and sell this software
* and its documentation for any purpose is hereby granted without fee,
* provided that the above copyright notice appear in all copies and
* that both that copyright notice and this permission notice appear
* in supporting documentation.  Silicon Graphics makes no
* representations about the suitability of this software for any
* purpose.  It is provided "as is" without express or implied warranty.
*
```

Copyright (c) 2009 Alexander Stepanov and Paul McJones

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. The authors make no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

# Namespaces

| Namespace | Description |
|---|---|
| Global | The global namespace contains intrinsic concepts. |
| System | Contains fundamental classes, functions and type definitions. |
| System.Collections | Contains collection classes and functions that operate on collections. |
| System.Concepts | Contains system library concepts. |
| System.IO | Contains classes and functions for doing input and output. |
| System.Meta | Contains instrinsic type predicate functions. |
| System.Security | Contains classes and functions for computing cryptographic hash functions. |
| System.Text | Contains classes and functions for manipulating text. |
| System.Threading | Contains classes and functions for controlling multiple threads of execution. |
| System.Unicode | Contains conversions between UTF-8, UTF-16 and UTF-32 strings and Unicode character classification functions. |

# 1 Global Namespace

The global namespace contains intrinsic concepts.

## 1.1 Concepts

| Concept | Description |
|---|---|
| Common<T, U> | Sets a requirement that types T and U have a common type. The common type is exposed as type parameter CommonType. |
| Convertible<T, U> | Sets a requirement that type T is implicitly convertible to type U. |
| Derived<T, U> | Sets a requirement that types T type and U are class types and class T is derived from type class U. |
| ExplicitlyConvertible<T, U> | Sets a requirement that type T is explicitly convertible to type U. |
| NonReferenceType<T> | Sets a requirement that type T is not a reference type. |
| Same<T, U> | Sets a requirement that type T is exatly the same type as U. |

### 1.1.1 Common<T, U> Concept

Sets a requirement that types T and U have a common type. The common type is exposed as type parameter CommonType.

#### Syntax

```
public concept Common<T, U>;
```

#### Remarks

For example common type for **int** and **double** is **double** because **int** can be implicitly converted to **double**.

### 1.1.2 Convertible<T, U> Concept

Sets a requirement that type T is implicitly convertible to type U.

#### Syntax

```
public concept Convertible<T, U>;
```

#### Remarks

For example **int** is implicitly convertible to **double** but not vice versa. More generally: conversions that do not lose information are implicit in Cmajor.

### 1.1.3 Derived<T, U> Concept

Sets a requirement that types T type and U are class types and class T is derived from type class U.

#### Syntax

```
public concept Derived<T, U>;
```

### 1.1.4 ExplicitlyConvertible<T, U> Concept

Sets a requirement that type T is explicitly convertible to type U.

#### Syntax

```
public concept ExplicitlyConvertible<T, U>;
```

#### Remarks

For example **double** is explicitly convertible to **int**. Explicit conversions need a **cast**.

### 1.1.5 NonReferenceType<T> Concept

Sets a requirement that type T is not a reference type.

#### Syntax

```
public concept NonReferenceType<T>;
```

### 1.1.6 Same<T, U> Concept

Sets a requirement that type T is exatly the same type as U.

#### Syntax

```
public concept Same<T, U>;
```

# 2 System Namespace

Contains fundamental classes, functions and type definitions.

Figures 2.1 and 2.2 contain the classes in this namespace.

Figure 2.1: Class Diagram 1: Basic Classes

Figure 2.2: Class Diagram 2: Function Objects

## 2.2 Classes

| Class | Description |
|---|---|
| BackInsertIterator<C> | An output iterator that inserts elements to the end of a back insertion sequence. |
| BackInsertProxy<C> | Implementation detail. |
| BinaryFun<Argument1, Argument2, Result> | A base class for binary function objects. |
| BinaryPred<Argument1, Argument2> | A base class for binary predicates. |
| BitAnd<T> | Bitwise AND binary function object. |
| BitNot<T> | Bitwise complement unary function object. |
| BitOr<T> | Bitwise inclusive OR binary function object. |
| BitXor<T> | Bitwise exclusive OR binary function object. |
| Console | A static class that contains console input and output functions. |
| ConversionException | An exception thrown when a conversion function fails. |
| Counter<T> | A counter class that stores a pointer to the counted object. |
| CounterBase | An abstract base class for SharedCount<T> and WeakCount<T> that keeps track of use count and weak count. |
| Date | A class for representing a date. |
| Divides<T> | Division binary function object. |
| Duration | Represents a duration in nanoseconds. |
| EndLine | Represents an end of line character. |
| EqualTo2<T, U> | An *equal to* binary predicate. |
| EqualTo<T> | An *equal to* relation. |
| Exception | A base class for all exception classes. |
| ExceptionPtr | The ExceptionPtr class can be used to hold an exception with full type information. |

| | |
|---|---|
| FrontInsertIterator<C> | An output iterator that inserts elements to the front of a container. |
| FrontInsertProxy<C> | Implementation detail. |
| Greater2<T, U> | A *greater than* binary predicate. |
| Greater<T> | A *greater than* relation. |
| GreaterOrEqualTo2<T, U> | A *greater than or equal to* binary predicate. |
| GreaterOrEqualTo<T> | A *greater than or equal to* relation. |
| Handle | A RAII class for storing a file descriptor. |
| Identity<T> | An identity unary function object. |
| InsertIterator<C> | An output iterator that inserts elements to given position of a container. |
| InsertProxy<C> | Implementation detail. |
| Less2<T, U> | A *less than* binary predicate. |
| Less<T> | A *less than* relation. |
| LessOrEqualTo2<T, U> | A *less than or equal to* binary predicate. |
| LessOrEqualTo<T> | A *less than or equal to* relation. |
| LogicalAnd<T> | A *logical AND* binary predicate. |
| LogicalNot<T> | A *logical NOT* unary predicate. |
| LogicalOr<T> | A *logical OR* binary predicate. |
| MT | Mersenne Twister pseudorandom number generator. |
| Minus<T> | A subtraction binary function object. |
| Multiplies<T> | A multiplication binary function object. |
| Negate<T> | A negation unary operation. |
| NotEqualTo2<T, U> | A *not equal to* binary predicate. |
| NotEqualTo<T> | An *not equal to* relation. |

| | |
|---|---|
| Pair<T, U> | A pair of values. |
| Plus<T> | An addition binary function object. |
| RandomAccessIter<T, R, P> | A random access iterator that contains a pointer to elements. |
| Rel<Argument> | A base class for relation function objects. |
| Remainder<T> | A remainder function object. |
| SelectFirst<T, U> | A function object for returning the first component of a pair. |
| SelectSecond<T, U> | A function object for returning the second component of a pair. |
| ShareableFromThis<T> | A class that implements the "shared from this" idiom. |
| SharedCount<T> | A handle to a Counter<T> that maintains the use count portion of the counter. |
| SharedPtr<T> | A shared pointer to an object. |
| ShiftLeft<T> | Shift left binary function. |
| ShiftRight<T> | Shift right binary function. |
| String<CharT> | A string class template. |
| TimeError | An exception thrown when a time function fails. |
| TimePoint | Represents a point in time as specified nanoseconds elapsed since epoch. |
| TracedFun | Function tracer. |
| Tracer | A utility class for tracing entry and exit of some operation. |
| UnaryFun<Argument, Result> | A base class for unary function objects. |
| UnaryPred<Argument> | A base class for unary predicates. |
| UniquePtr<T> | A unique pointer to an object. |

| | |
|---|---|
| WeakCount<T> | A handle to a Counter<T> that maintains the weak count portion of the counter. |
| WeakPtr<T> | Used to break cycles in shared ownership. |
| uhuge | 128-bit unsigned integer type. |

### 2.2.1 BackInsertIterator<C> Class

An output iterator that inserts elements to the end of a back insertion sequence.

#### Syntax

```
public class BackInsertIterator<C>;
```

#### Constraint

C is BackInsertionSequence

#### Model of

OutputIterator<T>

#### 2.2.1.1 Remarks

BackInserter(C&) is a helper function that returns a **BackInsertIterator<C>** for a back insertion sequence.

#### 2.2.1.2 Example

```
using System;
using System.Collections;

//  Writes:
//  1, 2, 3

void main()
{
    Set<int> s;
    s.Insert(2);
    s.Insert(3);
    s.Insert(1);
//  Set is a sorted container, so it contains now 1, 2, 3...

    List<int> list; // list is a model of a back insertion sequence

//  Copy ints from s to the end of list using BackInsertIterator...
    Copy(s.CBegin(), s.CEnd(), BackInserter(list));

//  ForwardContainers containing ints can be put to OutputStream...
    Console.Out() << list << endl();
}
```

#### 2.2.1.3 Type Definitions

| Name | Type | Description |
|---|---|---|
| PointerType | System.BackInsertProxy<C>* | Pointer to implementation defined proxy type. |
| ReferenceType | System.BackInsertProxy<C>& | Reference to implementation defined proxy type. |
| ValueType | System.BackInsertProxy<C> | Implementation defined proxy type. |

#### 2.2.1.4 Member Functions

| Member Function | Description |
|---|---|
| BackInsertIterator<C>() | Constructor. Default constructs a back insert iterator. |
| BackInsertIterator<C>(const System.-BackInsertIterator<C>&) | Copy constructor. |
| operator=(const System.BackInsertIterator<C>-&) | Copy assignment. |
| BackInsertIterator<C>(System.-BackInsertIterator<C>&&) | Move constructor. |
| operator=(System.BackInsertIterator<C>&&) | Move assignment. |
| BackInsertIterator<C>(C&) | Constructor. Constructs a back insert iterator with a container. |
| operator*() | Returns a reference to a proxy object that inserts values to the end of a container. |
| operator++() | Advances the iterator to the next element. |
| operator->() | Returns a pointer to a proxy object that inserts values to the end of a container. |

### BackInsertIterator<C>() Member Function

Constructor. Default constructs a back insert iterator.

#### Syntax

```
public BackInsertIterator<C>();
```

### BackInsertIterator<C>(const System.BackInsertIterator<C>&) Member Function

Copy constructor.

#### Syntax

```
public BackInsertIterator<C>(const System.BackInsertIterator<C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BackInsertIterator<C>& | Argument to copy. |

### operator=(const System.BackInsertIterator<C>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.BackInsertIterator<C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BackInsertIterator<C>& | Argument to assign. |

### BackInsertIterator<C>(System.BackInsertIterator<C>&&) Member Function

Move constructor.

#### Syntax

```
public BackInsertIterator<C>(System.BackInsertIterator<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BackInsertIterator<C>&& | Argument to move from. |

### operator=(System.BackInsertIterator<C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.BackInsertIterator<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BackInsertIterator<C>&& | Argument to assign from. |

### BackInsertIterator<C>(C&) Member Function

Constructor. Constructs a back insert iterator with a container.

#### Syntax

```
public BackInsertIterator<C>(C& c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | A container to which to insert elements. |

### operator*() Member Function

Returns a reference to a proxy object that inserts values to the end of a container.

### Syntax

```
public System.BackInsertProxy<C>& operator*();
```

### Returns

System.BackInsertProxy<C>&

Returns a reference to a proxy object that inserts values to the end of a container.

### operator++() Member Function

Advances the iterator to the next element.

#### Syntax

```
public System.BackInsertIterator<C>& operator++();
```

#### Returns

System.BackInsertIterator<C>&

Returns a reference to the iterator.

### operator->() Member Function

Returns a pointer to a proxy object that inserts values to the end of a container.

### Syntax

```
public System.BackInsertProxy<C>* operator->();
```

### Returns

System.BackInsertProxy<C>*

Returns a pointer to a proxy object that inserts values to the end of a container.

## 2.2.2 BackInsertProxy<C> Class

Implementation detail.

### Syntax

```
public class BackInsertProxy<C>;
```

### Constraint

C is BackInsertionSequence

#### 2.2.2.1 Member Functions

| Member Function | Description |
|---|---|
| BackInsertProxy<C>(System.-BackInsertProxy<C>&&) | Move constructor. |
| operator=(System.BackInsertProxy<C>&&) | Move assignment. |

### BackInsertProxy<C>(System.BackInsertProxy<C>&&) Member Function

Move constructor.

#### Syntax

```
public BackInsertProxy<C>(System.BackInsertProxy<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BackInsertProxy<C>&& | Argument to move from. |

### operator=(System.BackInsertProxy<C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.BackInsertProxy<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BackInsertProxy<C>&& | Argument to assign from. |

### 2.2.3 BinaryFun<Argument1, Argument2, Result> Class

A base class for binary function objects.

#### Syntax

```
public class BinaryFun<Argument1, Argument2, Result>;
```

#### Constraint

Argument1 is Semiregular and Argument2 is Semiregular

#### 2.2.3.1 Remarks

A derived binary function inherits the type definitions of this base class and provides an implementation for the *operator()(FirstArgumentType, SecondArgumentType)* function.

#### 2.2.3.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| FirstArgumentType | Argument1 | The type of the first argument of the binary function. |
| ResultType | Result | The type of the result of the binary function. |
| SecondArgumentType | Argument2 | The type of the second argument of the binary function. |

#### 2.2.3.3 Member Functions

| Member Function | Description |
|---|---|
| BinaryFun<Argument1, Argument2, Result>() | Default constructor. |
| BinaryFun<Argument1, Argument2, Result>-(const System.BinaryFun<Argument1, Argument2, Result>&) | Copy constructor. |
| operator=(const System.BinaryFun<Argument1, Argument2, Result>&) | Copy assignment. |
| BinaryFun<Argument1, Argument2, Result>-(System.BinaryFun<Argument1, Argument2, Result>&&) | Move constructor. |
| operator=(System.BinaryFun<Argument1, Argument2, Result>&&) | Move assignment. |

### BinaryFun<Argument1, Argument2, Result>() Member Function

Default constructor.

#### Syntax

```
public BinaryFun<Argument1, Argument2, Result>();
```

### BinaryFun<Argument1, Argument2, Result>(const System.BinaryFun<Argument1, Argument2, Result>&) Member Function

Copy constructor.

#### Syntax

```
public BinaryFun<Argument1, Argument2, Result>(const System.BinaryFun<Argument1,
Argument2, Result>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BinaryFun<Argument1, Argument2, Result>& | Argument to copy. |

### operator=(const System.BinaryFun<Argument1, Argument2, Result>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.BinaryFun<Argument1, Argument2, Result>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BinaryFun<Argument1, Argument2, Result>& | Argument to assign. |

### BinaryFun<Argument1, Argument2, Result>(System.BinaryFun<Argument1, Argument2, Result>&&) Member Function

Move constructor.

#### Syntax

```
public BinaryFun<Argument1, Argument2, Result>(System.BinaryFun<Argument1, Argument2,
Result>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BinaryFun<Argument1, Argument2, Result>-&& | Argument to move from. |

### operator=(System.BinaryFun<Argument1, Argument2, Result>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.BinaryFun<Argument1, Argument2, Result>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BinaryFun<Argument1, Argument2, Result>-&& | Argument to assign from. |

## 2.2.4 BinaryPred<Argument1, Argument2> Class

A base class for binary predicates.

### Syntax

```
public class BinaryPred<Argument1, Argument2>;
```

### Constraint

Argument1 is Semiregular and Argument2 is Semiregular

### Model of

BinaryPredicate<T>

### Base Class

System.BinaryFun<Argument1, Argument2, bool>

#### 2.2.4.1 Remarks

A binary predicate is a binary function whose application operator returns a truth value.

#### 2.2.4.2 Member Functions

| Member Function | Description |
|---|---|
| BinaryPred<Argument1, Argument2>() | Default constructor. |
| BinaryPred<Argument1, Argument2>(const System.BinaryPred<Argument1, Argument2>&) | Copy constructor. |
| operator=(const System.BinaryPred<-Argument1, Argument2>&) | Copy assignment. |
| BinaryPred<Argument1, Argument2>(System.-BinaryPred<Argument1, Argument2>&&) | Move constructor. |
| operator=(System.BinaryPred<Argument1, Argument2>&&) | Move assignment. |

### BinaryPred<Argument1, Argument2>() Member Function

Default constructor.

#### Syntax

```
public BinaryPred<Argument1, Argument2>();
```

### BinaryPred<Argument1, Argument2>(const System.BinaryPred<Argument1, Argument2>&) Member Function

Copy constructor.

#### Syntax

```
public BinaryPred<Argument1, Argument2>(const System.BinaryPred<Argument1, Argument2>&
that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BinaryPred<Argument1, Argument2>& | Argument to copy. |

### operator=(const System.BinaryPred<Argument1, Argument2>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.BinaryPred<Argument1, Argument2>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BinaryPred<Argument1, Argument2>& | Argument to assign. |

### BinaryPred<Argument1, Argument2>(System.BinaryPred<Argument1, Argument2>&&) Member Function

Move constructor.

#### Syntax

```
public BinaryPred<Argument1, Argument2>(System.BinaryPred<Argument1, Argument2>&&
that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BinaryPred<Argument1, Argument2>&& | Argument to move from. |

### operator=(System.BinaryPred<Argument1, Argument2>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.BinaryPred<Argument1, Argument2>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BinaryPred<Argument1, Argument2>&& | Argument to assign from. |

### 2.2.5 BitAnd<T> Class

Bitwise AND binary function object.

#### Syntax

```
public class BitAnd<T>;
```

#### Constraint

T is Semiregular

#### Base Class

System.BinaryFun<T, T, T>

#### 2.2.5.1 Member Functions

| Member Function | Description |
|---|---|
| BitAnd<T>() | Default constructor. |
| BitAnd<T>(const System.BitAnd<T>&) | Copy constructor. |
| operator=(const System.BitAnd<T>&) | Copy assignment. |
| BitAnd<T>(System.BitAnd<T>&&) | Move constructor. |
| operator=(System.BitAnd<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns bitwise AND of left and right. |

### BitAnd<T>() Member Function

Default constructor.

#### Syntax

```
public BitAnd<T>();
```

### BitAnd<T>(const System.BitAnd<T>&) Member Function

Copy constructor.

#### Syntax

```
public BitAnd<T>(const System.BitAnd<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BitAnd<T>& | Argument to copy. |

### operator=(const System.BitAnd<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.BitAnd<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BitAnd<T>& | Argument to assign. |

### BitAnd<T>(System.BitAnd<T>&&) Member Function

Move constructor.

#### Syntax

```
public BitAnd<T>(System.BitAnd<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BitAnd<T>&& | Argument to move from. |

### operator=(System.BitAnd<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.BitAnd<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BitAnd<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns bitwise AND of left and right.

#### Syntax

```
public T operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | Left argument. |
| right | const T& | Right argument. |

#### Returns

T

Returns bitwise AND of left and right.

## 2.2.6 BitNot<T> Class

Bitwise complement unary function object.

### Syntax

```
public class BitNot<T>;
```

### Constraint

T is Semiregular

### Base Class

System.UnaryFun<T, T>

#### 2.2.6.1 Member Functions

| Member Function | Description |
|---|---|
| BitNot<T>() | Default constructor. |
| BitNot<T>(const System.BitNot<T>&) | Copy constructor. |
| operator=(const System.BitNot<T>&) | Copy assignment. |
| BitNot<T>(System.BitNot<T>&&) | Move constructor. |
| operator=(System.BitNot<T>&&) | Move assignment. |
| operator()(const T&) const | Returns bitwise complement of operand. |

### BitNot<T>() Member Function

Default constructor.

#### Syntax

```
public BitNot<T>();
```

### BitNot<T>(const System.BitNot<T>&) Member Function

Copy constructor.

#### Syntax

```
public BitNot<T>(const System.BitNot<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BitNot<T>& | Argument to copy. |

### operator=(const System.BitNot<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.BitNot<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BitNot<T>& | Argument to assign. |

### BitNot<T>(System.BitNot<T>&&) Member Function

Move constructor.

#### Syntax

```
public BitNot<T>(System.BitNot<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BitNot<T>&& | Argument to move from. |

### operator=(System.BitNot<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.BitNot<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BitNot<T>&& | Argument to assign from. |

### operator()(const T&) const Member Function

Returns bitwise complement of operand.

#### Syntax

```
public T operator()(const T& operand) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| operand | const T& | Argument. |

#### Returns

T

Returns bitwise complement of operand.

## 2.2.7 BitOr<T> Class

Bitwise inclusive OR binary function object.

### Syntax

```
public class BitOr<T>;
```

### Constraint

T is Semiregular

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.7.1 Member Functions

| Member Function | Description |
|---|---|
| BitOr<T>() | Default constructor. |
| BitOr<T>(const System.BitOr<T>&) | Copy constructor. |
| operator=(const System.BitOr<T>&) | Copy assignment. |
| BitOr<T>(System.BitOr<T>&&) | Move constructor. |
| operator=(System.BitOr<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns bitwise inclusive OR of left and right. |

### BitOr<T>() Member Function

Default constructor.

#### Syntax

```
public BitOr<T>();
```

### BitOr<T>(const System.BitOr<T>&) Member Function

Copy constructor.

#### Syntax

```
public BitOr<T>(const System.BitOr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BitOr<T>& | Argument to copy. |

### operator=(const System.BitOr<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.BitOr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BitOr<T>& | Argument to assign. |

### BitOr<T>(System.BitOr<T>&&) Member Function

Move constructor.

#### Syntax

```
public BitOr<T>(System.BitOr<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BitOr<T>&& | Argument to move from. |

### operator=(System.BitOr<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.BitOr<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BitOr<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns bitwise inclusive OR of left and right.

#### Syntax

```
public T operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | Left argument. |
| right | const T& | Right argument. |

#### Returns

T

Returns bitwise inclusive OR of left and right.

## 2.2.8 BitXor<T> Class

Bitwise exclusive OR binary function object.

### Syntax

```
public class BitXor<T>;
```

### Constraint

T is Semiregular

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.8.1 Member Functions

| Member Function | Description |
|---|---|
| BitXor<T>() | Default constructor. |
| BitXor<T>(const System.BitXor<T>&) | Copy constructor. |
| operator=(const System.BitXor<T>&) | Copy assignment. |
| BitXor<T>(System.BitXor<T>&&) | Move constructor. |
| operator=(System.BitXor<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns bitwise exclusive OR of left and right. |

### BitXor<T>() Member Function

Default constructor.

#### Syntax

```
public BitXor<T>();
```

### BitXor<T>(const System.BitXor<T>&) Member Function

Copy constructor.

#### Syntax

```
public BitXor<T>(const System.BitXor<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BitXor<T>& | Argument to copy. |

### operator=(const System.BitXor<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.BitXor<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.BitXor<T>& | Argument to assign. |

### BitXor<T>(System.BitXor<T>&&) Member Function

Move constructor.

#### Syntax

```
public BitXor<T>(System.BitXor<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BitXor<T>&& | Argument to move from. |

### operator=(System.BitXor<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.BitXor<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.BitXor<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns bitwise exclusive OR of left and right.

#### Syntax

```
public T operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | Left argument. |
| right | const T& | Right argument. |

#### Returns

T

Returns bitwise exclusive OR of left and right.

## 2.2.9 Console Class

A static class that contains console input and output functions.

### Syntax

```
public static class Console;
```

#### 2.2.9.1 Remarks

By default **Console** reads from the standard input stream and writes to the standard output stream of the process. It also contains an error stream that by default refers to the standard error stream of the process.
The input, output and error streams that the **Console** class uses can be set with the SetIn(System.UniquePtr<System.IO.InputStream>&&), SetOut(System.UniquePtr<System.IO.OutputStream>&&) and SetError(System.UniquePtr<System.IO.OutputStream>&&) functions.

#### 2.2.9.2 Example

```
using System;

void main()
{
    Console.WriteLine("Hello, World!");

    Console.WriteLine(10);

    int lineNumber = 1234;
    Console.Error() << "Error in line " << lineNumber << endl();

    Console.WriteLine("What's your name?");
    Console.Write("> ");
    string name = Console.ReadLine();

    Console.Out() << "Hello " << name << '!' << endl();
}
```

#### 2.2.9.3 Member Functions

| Member Function | Description |
|---|---|
| Error() | Returns a reference to the console error stream. |
| In() | Returns a reference to the console input stream. |
| Out() | Returns a reference to the console output stream. |
| ReadLine() | Reads a line of text from the console input stream and returns it. |

| | |
|---|---|
| ReadToEnd() | Returns a string that contains the rest of input of the console input stream. |
| SetError(System.UniquePtr<System.IO.-OutputStream>&&) | Sets the error stream the **Console** class uses. |
| SetIn(System.UniquePtr<System.IO.-InputStream>&&) | Sets the input stream the **Console** class uses. |
| SetOut(System.UniquePtr<System.IO.-OutputStream>&&) | Sets the output stream the **Console** class uses. |
| Write(bool) | Writes a Boolean value to the console output stream. |
| Write(byte) | Writes a byte to the console output stream. |
| Write(char) | Writes a character to the console output stream. |
| Write(const System.String<char>&) const | Writes a string to the console output stream. |
| Write(const System.String<uchar>&) const | Writes a UTF-32 string to the console output stream. |
| Write(const System.String<wchar>&) const | Writes a UTF-16 string to console output stream. |
| Write(const char*) const | Writes a C-style string to the console output stream. |
| Write(double) | Writes a double value to the console output stream. |
| Write(float) | Writes a float value to the console output stream. |
| Write(int) | Writes an int value to the console output stream. |
| Write(long) | Writes a long value to the console output stream. |
| Write(sbyte) | Writes a signed byte to the console output stream. |
| Write(short) | Writes a short value to the console output stream |
| Write(uchar) | Writes a UTF-32 character to the console output stream. |
| Write(uint) | Writes an unsigned int to the console output stream. |
| Write(ulong) | Writes an unsigned long to the console output stream. |

| | |
|---|---|
| Write(ushort) | Writes an unsigned short to the console output stream. |
| Write(wchar) | Writers a UTF-16 character to the console output stream. |
| WriteLine() | Writes a newline to the console output stream. |
| WriteLine(bool) | Writes a Boolean value followed by a newline to the console output stream. |
| WriteLine(byte) | Writes a byte followed by a newline to the console output stream. |
| WriteLine(char) | Writes a character followed by a newline to the console output stream. |
| WriteLine(const System.String<char>&) const | Writes a string followed by a newline to the console output stream. |
| WriteLine(const System.String<uchar>&) const | Writes a UTF-32 string followed by a newline to the console output stream. |
| WriteLine(const System.String<wchar>&) const | Writes a UTF-16 string followed by a newline to the console output stream. |
| WriteLine(const char*) const | Writes a C-style string followed by a newline to the console output stream. |
| WriteLine(double) | Writes a double value followed by a newline to the console output stream. |
| WriteLine(float) | Writes a float value followed by a newline to the console output stream. |
| WriteLine(int) | Writes an int value followed by a newline to the console output stream. |
| WriteLine(long) | Writes a long value followed by a newline to the console output stream. |
| WriteLine(sbyte) | Writes a signed byte followed by a newline to the console output stream. |
| WriteLine(short) | Writes a short value followed by a newline to the console output stream. |

| | |
|---|---|
| WriteLine(uchar) | Writes a UTF-32 character followed by a newline to the console output stream. |
| WriteLine(uint) | Writes a unsigned int followed by a newline to the console output stream. |
| WriteLine(ulong) | Writes an unsigned long followed by a newline to the console output stream. |
| WriteLine(ushort) | Writes an unsigned short followed by a newline to the console output stream. |
| WriteLine(wchar) | Writes a UTF-16 character followed by a newline to the console output stream. |

### Error() Member Function

Returns a reference to the console error stream.

### Syntax

```
public static System.IO.OutputStream& Error();
```

### Returns

System.IO.OutputStream&

Returns a reference to the console error stream.

### In() Member Function

Returns a reference to the console input stream.

### Syntax

```
public static System.IO.InputStream& In();
```

### Returns

System.IO.InputStream&

Returns a reference to the console input stream.

### Out() Member Function

Returns a reference to the console output stream.

#### Syntax

```
public static System.IO.OutputStream& Out();
```

#### Returns

System.IO.OutputStream&

Returns a reference to the console output stream.

### ReadLine() Member Function

Reads a line of text from the console input stream and returns it.

#### Syntax

```
public static System.String<char> ReadLine();
```

#### Returns

System.String<char>

Returns a line of text read from the console input stream.

### ReadToEnd() Member Function

Returns a string that contains the rest of input of the console input stream.

### Syntax

```
public static System.String<char> ReadToEnd();
```

### Returns

System.String<char>

Returns a string that contains the rest of input of the console input stream.

### SetError(System.UniquePtr<System.IO.OutputStream>&&) Member Function

Sets the error stream the Console class uses.

#### Syntax

```
public static void SetError(System.UniquePtr<System.IO.OutputStream>&& err_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| err_ | System.UniquePtr<System.IO.OutputStream>&& | A unique pointer to an output stream. |

### SetIn(System.UniquePtr<System.IO.InputStream>&&) Member Function

Sets the input stream the Console class uses.

#### Syntax

```
public static void SetIn(System.UniquePtr<System.IO.InputStream>&& in_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| in_ | System.UniquePtr<System.IO.InputStream>&& | A unique pointer to an input stream. |

### SetOut(System.UniquePtr<System.IO.OutputStream>&&) Member Function

Sets the output stream the Console class uses.

#### Syntax

```
public static void SetOut(System.UniquePtr<System.IO.OutputStream>&& out_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| out_ | System.UniquePtr<System.IO.OutputStream>&& | A unique pointer to an output stream. |

#### Example

```
using System;
using System.IO;

void main()
{
    try
    {
        Console.SetOut(UniquePtr<OutputStream>(new OutputFileStream("trace.
            log")));
        Console.WriteLine("this goes to trace.log");
    }
    catch (const Exception& ex)
    {
        Console.Error() << ex.ToString() << endl();
    }
}
```

### Write(bool) Member Function

Writes a Boolean value to the console output stream.

#### Syntax

```
public static void Write(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean value to write. |

### Write(byte) Member Function

Writes a byte to the console output stream.

#### Syntax

```
public static void Write(byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A byte to write. |

### Write(char) Member Function

Writes a character to the console output stream.

#### Syntax

```
public static void Write(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### Write(const System.String<char>&) const Member Function

Writes a string to the console output stream.

#### Syntax

```
public static void Write(const System.String<char>& s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to write. |

### Write(const System.String<uchar>&) const Member Function

Writes a UTF-32 string to the console output stream.

#### Syntax

```
public static void Write(const System.String<uchar>& s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to write. |

#### Remarks

In Linux the string is written UTF-8 encoded. In Windows the string is written UTF-16 encoded.

### Write(const System.String<wchar>&) const Member Function

Writes a UTF-16 string to console output stream.

#### Syntax

```
public static void Write(const System.String<wchar>& s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to write. |

#### Remarks

In Linux the string is written UTF-8 encoded. In Windows the string is written UTF-16 encoded.

### Write(const char*) const Member Function

Writes a C-style string to the console output stream.

#### Syntax

```
public static void Write(const char* s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### Write(double) Member Function

Writes a double value to the console output stream.

#### Syntax

```
public static void Write(double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A double value to write. |

### Write(float) Member Function

Writes a float value to the console output stream.

#### Syntax

```
public static void Write(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A float value to write. |

### Write(int) Member Function

Writes an int value to the console output stream.

#### Syntax

```
public static void Write(int i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An int value to write. |

### Write(long) Member Function

Writes a long value to the console output stream.

#### Syntax

```
public static void Write(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A long to write. |

### Write(sbyte) Member Function

Writes a signed byte to the console output stream.

#### Syntax

```
public static void Write(sbyte s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | sbyte | A signed byte to write. |

### Write(short) Member Function

Writes a short value to the console output stream

#### Syntax

```
public static void Write(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A short value to write. |

### Write(uchar) Member Function

Writes a UTF-32 character to the console output stream.

#### Syntax

```
public static void Write(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A character to write. |

#### Remarks

In Linux the character is written as a sequence of UTF-8 encoded bytes. In Windows the character is written as UTF-16 encoded.

### Write(uint) Member Function

Writes an unsigned int to the console output stream.

#### Syntax

```
public static void Write(uint u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | uint | An unsigned int to write. |

### Write(ulong) Member Function

Writes an unsigned long to the console output stream.

#### Syntax

```
public static void Write(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | An unsigned long to write. |

### Write(ushort) Member Function

Writes an unsigned short to the console output stream.

#### Syntax

```
public static void Write(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | An unsigned short to write. |

### Write(wchar) Member Function

Writers a UTF-16 character to the console output stream.

#### Syntax

```
public static void Write(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A character to write. |

#### Remarks

In Linux the character is written as a sequence of UTF-8 encoded bytes. In Windows the character is written as UTF-16 encoded.

### WriteLine() Member Function

Writes a newline to the console output stream.

### Syntax

```
public static void WriteLine();
```

### WriteLine(bool) Member Function

Writes a Boolean value followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean value to write. |

### WriteLine(byte) Member Function

Writes a byte followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A byte to write. |

### WriteLine(char) Member Function

Writes a character followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### WriteLine(const System.String<char>&) const Member Function

Writes a string followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(const System.String<char>& s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to write. |

### WriteLine(const System.String<uchar>&) const Member Function

Writes a UTF-32 string followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(const System.String<uchar>& s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to write. |

#### Remarks

In Linux the string is written UTF-8 encoded. In Windows the string is written UTF-16 encoded.

### WriteLine(const System.String<wchar>&) const Member Function

Writes a UTF-16 string followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(const System.String<wchar>& s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to write. |

#### Remarks

In Linux the string is written UTF-8 encoded. In Windows the string is written UTF-16 encoded.

### WriteLine(const char*) const Member Function

Writes a C-style string followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(const char* s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### WriteLine(double) Member Function

Writes a double value followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A double value to write. |

### WriteLine(float) Member Function

Writes a float value followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A float value to write. |

### WriteLine(int) Member Function

Writes an int value followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(int i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An int value to write. |

### WriteLine(long) Member Function

Writes a long value followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A long value to write. |

### WriteLine(sbyte) Member Function

Writes a signed byte followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(sbyte s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | sbyte | A signed byte to write. |

### WriteLine(short) Member Function

Writes a short value followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A short value to write. |

### WriteLine(uchar) Member Function

Writes a UTF-32 character followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A character to write. |

#### Remarks

In Linux the character is written as a sequence of UTF-8 encoded bytes. In Windows the character is written as UTF-16 encoded.

### WriteLine(uint) Member Function

Writes a unsigned int followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(uint u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | uint | An unsigned int to write. |

### WriteLine(ulong) Member Function

Writes an unsigned long followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | An unsigned long to write. |

### WriteLine(ushort) Member Function

Writes an unsigned short followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | An unsigned short to write. |

### WriteLine(wchar) Member Function

Writes a UTF-16 character followed by a newline to the console output stream.

#### Syntax

```
public static void WriteLine(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A character to write. |

#### Remarks

In Linux the character is written as a sequence of UTF-8 encoded bytes. In Windows the character is written as UTF-16 encoded.

## 2.2.10 ConversionException Class

An exception thrown when a conversion function fails.

### Syntax

```
public class ConversionException;
```

### Base Class

System.Exception

#### 2.2.10.1 Remarks

Conversion functions that throw **ConversionException** are: System.ParseBool.C.R.System.String, System.ParseDate.C.R.System.String, System.ParseDouble.C.R.System.String System.ParseHex.C.R.System.String, System.ParseHexUHuge.C.R.System.String, System.ParseInt.C.R.System.String, System.ParseUInt.C.R.System.String, System.ParseUHuge.C.R.System.String and System.ParseULong.C.R.System.String.

#### 2.2.10.2 Example

```
using System;

//  Writes:
//  integer value cannot be parsed from input string '123.456'

void main()
{
    try
    {
        int x = ParseInt("123.456");
    }
    catch (const ConversionException& ex)
    {
        Console.WriteLine(ex.Message());
    }
}
```

#### 2.2.10.3 Member Functions

| Member Function | Description |
|---|---|
| ConversionException() | Default constructor. |
| ConversionException(const System.-ConversionException&) | Copy constructor. |
| operator=(const System.ConversionException&) | Copy assignment. |
| ConversionException(System.-ConversionException&&) | Move constructor. |

| | |
|---|---|
| operator=(System.ConversionException&&) | Move assignment. |

### ConversionException() Member Function

Default constructor.

#### Syntax

```
public ConversionException();
```

### ConversionException(const System.ConversionException&) Member Function

Copy constructor.

#### Syntax

```
public ConversionException(const System.ConversionException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.ConversionException& | Argument to copy. |

### operator=(const System.ConversionException&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.ConversionException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.ConversionException& | Argument to assign. |

### ConversionException(System.ConversionException&&) Member Function

Move constructor.

#### Syntax

```
public ConversionException(System.ConversionException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ConversionException&& | Argument to move from. |

### operator=(System.ConversionException&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.ConversionException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ConversionException&& | Argument to assign from. |

## 2.2.11 Counter<T> Class

A counter class that stores a pointer to the counted object.

### Syntax

```
public class Counter<T>;
```

### Base Class

System.CounterBase

#### 2.2.11.1 Remarks

Base class for SharedCount<T> and WeakCount<T>.

#### 2.2.11.2 Member Functions

| Member Function | Description |
|---|---|
| Counter<T>() | Default constructor. |
| Counter<T>(T*) | Constructor. Stores a pointer to counted object. |
| Dispose() | Overridden. Deletes the counted object. |

### Counter<T>() Member Function

Default constructor.

### Syntax

```
public Counter<T>();
```

### Counter<T>(T*) Member Function

Constructor. Stores a pointer to counted object.

#### Syntax

```
public Counter<T>(T* ptr_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to counted object. |

### Dispose() Member Function

Overridden. Deletes the counted object.

### Syntax

```
public void Dispose();
```

## 2.2.12 CounterBase Class

An abstract base class for SharedCount<T> and WeakCount<T> that keeps track of use count and weak count.

### Syntax

```
public abstract class CounterBase;
```

### 2.2.12.1 Member Functions

| Member Function | Description |
|---|---|
| CounterBase() | Constructor. Sets use count and weak count to one. |
| AddReference() | Increments use count and weak count. |
| Destruct() | Deletes this counter. |
| Dispose() | Abstract disposing function overridden in Counter<T> class. |
| GetUseCount() const | Returns the use count. |
| Release() | Decrements use count and weak count. |
| WeakAddReference() | Increments weak count. |
| WeakRelease() | Decrements weak count. |
| ~CounterBase() | Destructor. |

### CounterBase() Member Function

Constructor. Sets use count and weak count to one.

#### Syntax

```
public CounterBase();
```

### AddReference() Member Function

Increments use count and weak count.

#### Syntax

```
public void AddReference();
```

### Destruct() Member Function

Deletes this counter.

### Syntax

```
public void Destruct();
```

### Dispose() Member Function

Abstract disposing function overridden in Counter<T> class.

### Syntax

```
public abstract void Dispose();
```

### GetUseCount() const Member Function

Returns the use count.

#### Syntax

```
public int GetUseCount() const;
```

#### Returns

int

Returns the use count.

### Release() Member Function

Decrements use count and weak count.

### Syntax

```
public void Release();
```

### Remarks

If use count has gone to zero, deletes the counted object by calling System.CounterBase.Dispose function. If the weak count also has gone to zero, deletes the counter object.

### WeakAddReference() Member Function

Increments weak count.

#### Syntax

```
public void WeakAddReference();
```

### WeakRelease() Member Function

Decrements weak count.

### Syntax

```
public void WeakRelease();
```

### Remarks

If weak count has gone to zero, calls System.CounterBase.Destruct to delete this counter object.

### ~CounterBase() Member Function

Destructor.

#### Syntax

```
public ~CounterBase();
```

## 2.2.13 Date Class

A class for representing a date.

### Syntax

```
public class Date;
```

#### 2.2.13.1 Remarks

CurrentDate() function returns current date.

#### 2.2.13.2 Member Functions

| Member Function | Description |
|---|---|
| Date() | Default constructor. Constructs a date 1.1.1. |
| Date(const System.Date&) | Copy constructor. |
| operator=(const System.Date&) | Copy assignment. |
| Date(System.Date&&) | Move constructor. |
| operator=(System.Date&&) | Move assignment. |
| Date(ushort, byte, byte) | Constructor. Initializes a date with given year, month and day. |
| Day() const | Returns the day part of the date. |
| Month() const | Returns the month part of the date. |
| Year() const | Returns the year part of the date. |

### Date() Member Function

Default constructor. Constructs a date 1.1.1.

### Syntax

```
public Date();
```

### Date(const System.Date&) Member Function

Copy constructor.

#### Syntax

```
public Date(const System.Date& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Date& | Argument to copy. |

### operator=(const System.Date&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Date& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Date& | Argument to assign. |

### Date(System.Date&&) Member Function

Move constructor.

#### Syntax

```
public Date(System.Date&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Date&& | Argument to move from. |

### operator=(System.Date&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Date&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Date&& | Argument to assign from. |

### Date(ushort, byte, byte) Member Function

Constructor. Initializes a date with given year, month and day.

#### Syntax

```
public Date(ushort year_, byte month_, byte day_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| year_ | ushort | Year. |
| month_ | byte | Month. |
| day_ | byte | Day. |

### Day() const Member Function

Returns the day part of the date.

#### Syntax

```
public byte Day() const;
```

#### Returns

byte

Returns the day part of the date.

### Month() const Member Function

Returns the month part of the date.

#### Syntax

```
public byte Month() const;
```

#### Returns

byte

Returns the month part of the date.

#### Year() const Member Function

Returns the year part of the date.

#### Syntax

```
public ushort Year() const;
```

#### Returns

ushort

Returns the year part of the date.

### 2.2.13.3 Nonmember Functions

| Function | Description |
|---|---|
| operator<(System.Date, System.Date) | Compares two dates for less than relationship. |
| operator==(System.Date, System.Date) | Compares two dates for equality. |

### operator<(System.Date, System.Date) Function

Compares two dates for less than relationship.

#### Syntax

```
public bool operator<(System.Date left, System.Date right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Date | The first date. |
| right | System.Date | The second date. |

#### Returns

bool

Returns true if the first date comes before the second date, false otherwise.

### operator==(System.Date, System.Date) Function

Compares two dates for equality.

#### Syntax

```
public bool operator==(System.Date left, System.Date right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Date | The first date. |
| right | System.Date | The second date. |

#### Returns

bool

Returns true, if the first date is equal to the second date, false otherwise.

## 2.2.14 Divides<T> Class

Division binary function object.

### Syntax

```
public class Divides<T>;
```

### Constraint

T is MultiplicativeGroup

### Model of

BinaryOperation<T>

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.14.1 Example

```
using System;
using System.Collections;

//  Writes:
//  2

void main()
{
    List<double> divisors;
    divisors.Add(2.0);
    divisors.Add(3.0);
    divisors.Add(4.0);
    double two = Accumulate(divisors.CBegin(), divisors.CEnd(), 48.0,
        Divides<double>());
    Console.WriteLine(two);
}
```

#### 2.2.14.2 Member Functions

| Member Function | Description |
|---|---|
| Divides<T>() | Default constructor. |
| Divides<T>(const System.Divides<T>&) | Copy constructor. |
| operator=(const System.Divides<T>&) | Copy assignment. |
| Divides<T>(System.Divides<T>&&) | Move constructor. |
| operator=(System.Divides<T>&&) | Move assignment. |

| | |
|---|---|
| operator()(const T&, const T&) const | Returns the first argument divided by the second argument. |

### Divides<T>() Member Function

Default constructor.

### Syntax

```
public Divides<T>();
```

### Divides<T>(const System.Divides<T>&) Member Function

Copy constructor.

#### Syntax

```
public Divides<T>(const System.Divides<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Divides<T>& | Argument to copy. |

### operator=(const System.Divides<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Divides<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Divides<T>& | Argument to assign. |

### Divides<T>(System.Divides<T>&&) Member Function

Move constructor.

#### Syntax

```
public Divides<T>(System.Divides<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Divides<T>&& | Argument to move from. |

### operator=(System.Divides<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Divides<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Divides<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns the first argument divided by the second argument.

#### Syntax

```
public T operator()(const T& a, const T& b) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | Dividend. |
| b | const T& | Divisor. |

#### Returns

T

Returns $a/b$.

## 2.2.15 Duration Class

Represents a duration in nanoseconds.

### Syntax

```
public class Duration;
```

#### 2.2.15.1 Member Functions

| Member Function | Description |
|---|---|
| Duration() | Default constructor. Initializes the duration to zero nanoseconds. |
| Duration(const System.Duration&) | Copy constructor. |
| operator=(const System.Duration&) | Copy assignment. |
| Duration(System.Duration&&) | Move constructor. |
| operator=(System.Duration&&) | Move assignment. |
| Duration(long) | Constructor. Initializes the duration to specified number of nanoseconds. |
| FromHours(long) | Returns a duration of specified number of hours. |
| FromMicroseconds(long) | Returns duration of specified number of microseconds. |
| FromMilliseconds(long) | Returns duration of specified number of millseconds. |
| FromMinutes(long) | Returns duration of specified number of minutes. |
| FromNanoseconds(long) | Returns duration of specified number of nanoseconds. |
| FromSeconds(long) | Returns duration of specified number of seconds. |
| Hours() const | Returns total hours elapsed. |
| Microseconds() const | Returns total microseconds elapsed. |
| Milliseconds() const | Returns total milliseconds elapsed. |
| Minutes() const | Returns total minutes elapsed. |
| Nanoseconds() const | Returns total nanoseconds elapsed. |

| | |
|---|---|
| Rep() const | Returns total nanoseconds elapsed. |
| Seconds() const | Returns total seconds elapsed. |

### Duration() Member Function

Default constructor. Initializes the duration to zero nanoseconds.

### Syntax

```
public Duration();
```

### Duration(const System.Duration&) Member Function

Copy constructor.

#### Syntax

```
public Duration(const System.Duration& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Duration& | Argument to copy. |

### operator=(const System.Duration&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Duration& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Duration& | Argument to assign. |

### Duration(System.Duration&&) Member Function

Move constructor.

#### Syntax

```
public Duration(System.Duration&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Duration&& | Argument to move from. |

### operator=(System.Duration&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Duration&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Duration&& | Argument to assign from. |

### Duration(long) Member Function

Constructor. Initializes the duration to specified number of nanoseconds.

#### Syntax

```
public Duration(long nanosecs_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| nanosecs_ | long | Nanoseconds. |

### FromHours(long) Member Function

Returns a duration of specified number of hours.

#### Syntax

```
public static System.Duration FromHours(long hours);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| hours | long | Hours. |

#### Returns

System.Duration

Returns a duration of specified number of hours.

### FromMicroseconds(long) Member Function

Returns duration of specified number of microseconds.

#### Syntax

```
public static System.Duration FromMicroseconds(long microseconds);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| microseconds | long | Microseconds. |

#### Returns

System.Duration

Returns duration of specified number of microseconds.

### FromMilliseconds(long) Member Function

Returns duration of specified number of millseconds.

#### Syntax

```
public static System.Duration FromMilliseconds(long milliseconds);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| milliseconds | long | Milliseconds. |

#### Returns

System.Duration

Returns duration of specified number of milliseconds.

### FromMinutes(long) Member Function

Returns duration of specified number of minutes.

#### Syntax

```
public static System.Duration FromMinutes(long minutes);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| minutes | long | Minutes. |

#### Returns

System.Duration

Returns duration of specified number of minutes.

### FromNanoseconds(long) Member Function

Returns duration of specified number of nanoseconds.

#### Syntax

```
public static System.Duration FromNanoseconds(long nanoseconds);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| nanoseconds | long | Nanoseconds. |

#### Returns

System.Duration

Returns duration of specified number of nanoseconds.

### FromSeconds(long) Member Function

Returns duration of specified number of seconds.

#### Syntax

```
public static System.Duration FromSeconds(long seconds);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| seconds | long | Seconds. |

#### Returns

System.Duration

Returns duration of specified number of seconds.

### Hours() const Member Function

Returns total hours elapsed.

### Syntax

```
public long Hours() const;
```

### Returns

long

Returns total hours elapsed.

### Microseconds() const Member Function

Returns total microseconds elapsed.

#### Syntax

```
public long Microseconds() const;
```

#### Returns

long

Returns total microseconds elapsed.

### Milliseconds() const Member Function

Returns total milliseconds elapsed.

#### Syntax

```
public long Milliseconds() const;
```

#### Returns

long

Returns total milliseconds elapsed.

### Minutes() const Member Function

Returns total minutes elapsed.

#### Syntax

```
public long Minutes() const;
```

#### Returns

long

Returns total minutes elapsed.

### Nanoseconds() const Member Function

Returns total nanoseconds elapsed.

#### Syntax

```
public long Nanoseconds() const;
```

#### Returns

long

Returns total nanoseconds elapsed.

### Rep() const Member Function

Returns total nanoseconds elapsed.

### Syntax

```
public long Rep() const;
```

### Returns

long

Returns total nanoseconds elapsed.

**Seconds() const Member Function**

Returns total seconds elapsed.

**Syntax**

```
public long Seconds() const;
```

**Returns**

long

Returns total seconds elapsed.

### 2.2.15.2 Nonmember Functions

| Function | Description |
|---|---|
| operator%(System.Duration, System.Duration) | Computes the remainder of division of two durations. |
| operator*(System.Duration, System.Duration) | Computes the product of two durations. |
| operator+(System.Duration, System.Duration) | Computes the sum of two durations. |
| operator+(System.Duration, System.TimePoint) | Computes the sum of a duration and a time point and returns a time point. |
| operator-(System.Duration, System.Duration) | Subtracts a duration from a duration. |
| operator/(System.Duration, System.Duration) | Division of two durations. |
| operator<(System.Duration, System.Duration) | Compares two durations for less than relationship. |
| operator==(System.Duration, System.Duration) | Compares two durations for equality. |

### operator%(System.Duration, System.Duration) Function

Computes the remainder of division of two durations.

#### Syntax

```
public System.Duration operator%(System.Duration left, System.Duration right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Duration | The first duration. |
| right | System.Duration | The second duration. |

#### Returns

System.Duration

Returns $left\%right$.

### operator*(System.Duration, System.Duration) Function

Computes the product of two durations.

#### Syntax

```
public System.Duration operator*(System.Duration left, System.Duration right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Duration | The first duration. |
| right | System.Duration | The second duration. |

#### Returns

System.Duration

Returns $left \times right$.

### operator+(System.Duration, System.Duration) Function

Computes the sum of two durations.

#### Syntax

```
public System.Duration operator+(System.Duration left, System.Duration right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Duration | The first duration. |
| right | System.Duration | The second duration. |

#### Returns

System.Duration

Returns $left + right$.

### operator+(System.Duration, System.TimePoint) Function

Computes the sum of a duration and a time point and returns a time point.

#### Syntax

```
public System.TimePoint operator+(System.Duration d, System.TimePoint tp);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | System.Duration | A duration. |
| tp | System.TimePoint | A time point. |

#### Returns

System.TimePoint

Returns $d + tp$.

### operator-(System.Duration, System.Duration) Function

Subtracts a duration from a duration.

#### Syntax

```
public System.Duration operator-(System.Duration left, System.Duration right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Duration | The first duration. |
| right | System.Duration | The second duration. |

#### Returns

System.Duration

Returns $left - right$.

### operator/(System.Duration, System.Duration) Function

Division of two durations.

#### Syntax

```
public System.Duration operator/(System.Duration left, System.Duration right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Duration | Dividend. |
| right | System.Duration | Divisor. |

#### Returns

System.Duration

Returns $left$ divided by $right$.

### operator<(System.Duration, System.Duration) Function

Compares two durations for less than relationship.

#### Syntax

```
public bool operator<(System.Duration left, System.Duration right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Duration | The first duration. |
| right | System.Duration | The second duration. |

#### Returns

bool

Returns true, if the first duration is less than the second duration, false otherwise.

### operator==(System.Duration, System.Duration) Function

Compares two durations for equality.

#### Syntax

```
public bool operator==(System.Duration left, System.Duration right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Duration | The first duration. |
| right | System.Duration | The second duration. |

#### Returns

bool

Returns true, if the first duration is equal to the second duration, false otherwise.

## 2.2.16 EndLine Class

Represents an end of line character.

### Syntax

```
public class EndLine;
```

### 2.2.16.1 Remarks

endl() function returns this for dispatching to operator<<(System.IO.OutputStream&, System.EndLine) function.

### 2.2.16.2 Member Functions

| Member Function | Description |
|---|---|
| EndLine() | Default constructor. |
| EndLine(const System.EndLine&) | Copy constructor. |
| EndLine(const System.EndLine&) | Copy constructor. |

### EndLine() Member Function

Default constructor.

### Syntax

```
public EndLine();
```

### EndLine(const System.EndLine&) Member Function

Copy constructor.

#### Syntax

```
public EndLine(const System.EndLine& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.EndLine& | |

### EndLine(const System.EndLine&) Member Function

Copy constructor.

#### Syntax

```
public EndLine(const System.EndLine& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.EndLine& | Argument to copy from. |

## 2.2.17 EqualTo2<T, U> Class

An *equal to* binary predicate.

### Syntax

```
public class EqualTo2<T, U>;
```

### Constraint

EqualityComparable<T, U>

### Model of

Relation<T, U, V>

### Base Class

System.BinaryPred<T, U>

#### 2.2.17.1 Remarks

T and U are possibly different types, but can be compared for equality.

#### 2.2.17.2 Member Functions

| Member Function | Description |
|---|---|
| EqualTo2<T, U>() | Default constructor. |
| EqualTo2<T, U>(const System.EqualTo2<T, U>&) | Copy constructor. |
| operator=(const System.EqualTo2<T, U>&) | Copy assignment. |
| EqualTo2<T, U>(System.EqualTo2<T, U>&&) | Move constructor. |
| operator=(System.EqualTo2<T, U>&&) | Move assignment. |
| operator()(const T&, const U&) const | Returns true if the first argument is equal to the second argument, false otherwise. |

### EqualTo2<T, U>() Member Function

Default constructor.

#### Syntax

```
public EqualTo2<T, U>();
```

### EqualTo2<T, U>(const System.EqualTo2<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public EqualTo2<T, U>(const System.EqualTo2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.EqualTo2<T, U>& | Argument to copy. |

### operator=(const System.EqualTo2<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.EqualTo2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.EqualTo2<T, U>& | Argument to assign. |

### EqualTo2<T, U>(System.EqualTo2<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public EqualTo2<T, U>(System.EqualTo2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.EqualTo2<T, U>&& | Argument to move from. |

### operator=(System.EqualTo2<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.EqualTo2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.EqualTo2<T, U>&& | Argument to assign from. |

### operator()(const T&, const U&) const Member Function

Returns true if the first argument is equal to the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const U& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const U& | The second argument. |

#### Returns

bool

Returns $left == right$.

## 2.2.18 EqualTo<T> Class

An *equal to* relation.

### Syntax

```
public class EqualTo<T>;
```

### Constraint

T is Regular

### Model of

Relation<T>

### Base Class

System.Rel<T>

#### 2.2.18.1 Member Functions

| Member Function | Description |
|---|---|
| EqualTo<T>() | Default constructor. |
| EqualTo<T>(const System.EqualTo<T>&) | Copy constructor. |
| operator=(const System.EqualTo<T>&) | Copy assignment. |
| EqualTo<T>(System.EqualTo<T>&&) | Move constructor. |
| operator=(System.EqualTo<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns true if the first argument is equal to the second argument, false otherwise. |

### EqualTo<T>() Member Function

Default constructor.

### Syntax

```
public EqualTo<T>();
```

### EqualTo<T>(const System.EqualTo<T>&) Member Function

Copy constructor.

#### Syntax

```
public EqualTo<T>(const System.EqualTo<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.EqualTo<T>& | Argument to copy. |

### operator=(const System.EqualTo<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.EqualTo<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.EqualTo<T>& | Argument to assign. |

### EqualTo<T>(System.EqualTo<T>&&) Member Function

Move constructor.

#### Syntax

```
public EqualTo<T>(System.EqualTo<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.EqualTo<T>&& | Argument to move from. |

### operator=(System.EqualTo<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.EqualTo<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.EqualTo<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns true if the first argument is equal to the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const T& | The second argument. |

#### Returns

bool

Returns $left == right$.

## 2.2.19 Exception Class

A base class for all exception classes.

### Syntax

```
public class Exception;
```

#### 2.2.19.1 Remarks

All possible Cmajor exceptions can be catched by catching **Exception** class.

#### 2.2.19.2 Example

```
using System;
using System.IO;

//  if file 'nonexistent.file' does not exist, writes a message like
//  "System.IO.OpenFileException at 'C:/Programming/cmajor++/system/
    filestream.cm' line 105:
//  could not open file 'nonexistent.file' for reading: No such file or
    directory"
//  to the standard error stream.

void main()
{
    try
    {
        InputFileStream foo("nonexistent.file");
    }
    catch (const Exception& ex)
    {
        Console.Error() << ex.ToString() << endl();
    }
}
```

#### 2.2.19.3 Member Functions

| Member Function | Description |
|---|---|
| Exception() | Default constructor. |
| Exception(const System.Exception&) | Copy constructor. |
| operator=(const System.Exception&) | Copy assignment. |
| Exception(System.Exception&&) | Move constructor. |
| operator=(System.Exception&&) | Move assignment. |
| Exception(const System.String<char>&) | Constructor. Constructs an exception with an error message. |

| | |
|---|---|
| ExceptionType() const | Returns the dynamic class name of the exception. |
| File() const | Returns the path to the source file where the exception was thrown. |
| Line() const | Returns the source line number where the exception was thrown. |
| Message() const | Returns the error message. |
| SetCallStack(const System.String<char>&) | Sets the call stack string. |
| SetExceptionType(const System.String<char>&-) | Sets the dynamic class name. |
| SetFile(const System.String<char>&) | Sets the source file. |
| SetLine(int) | Sets the source line number. |
| ToString() const | Returns the error message with the full name of the thrown exception class, and source file path and source line number where the exception was thrown. |
| ~Exception() | Destructor. |

### Exception() Member Function

Default constructor.

### Syntax

```
public Exception();
```

### Exception(const System.Exception&) Member Function

Copy constructor.

#### Syntax

```
public Exception(const System.Exception& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Exception& | An exception to copy from. |

### operator=(const System.Exception&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Exception& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Exception& | An exception to assign from. |

### Exception(System.Exception&&) Member Function

Move constructor.

#### Syntax

```
public Exception(System.Exception&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.Exception&& | An exception to move from. |

### operator=(System.Exception&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Exception&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.Exception&& | An exception to move from. |

### Exception(const System.String<char>&) Member Function

Constructor. Constructs an exception with an error message.

#### Syntax

```
public Exception(const System.String<char>& message_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| message_ | const System.String<char>& | An error message. |

### ExceptionType() const Member Function

Returns the dynamic class name of the exception.

#### Syntax

```
public const System.String<char>& ExceptionType() const;
```

#### Returns

const System.String<char>&

Returns the dynamic class name of the exception.

### File() const Member Function

Returns the path to the source file where the exception was thrown.

#### Syntax

```
public const System.String<char>& File() const;
```

#### Returns

const System.String<char>&

Returns the path to the source file where the exception was thrown.

### Line() const Member Function

Returns the source line number where the exception was thrown.

### Syntax

```
public int Line() const;
```

### Returns

int

Returns the source line number where the exception was thrown.

### Message() const Member Function

Returns the error message.

#### Syntax

```
public const System.String<char>& Message() const;
```

#### Returns

const System.String<char>&

Returns the error message.

### SetCallStack(const System.String<char>&) Member Function

Sets the call stack string.

#### Syntax

```
public void SetCallStack(const System.String<char>& callStack_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| callStack_ | const System.String<char>& | A call stack string. |

### SetExceptionType(const System.String<char>&) Member Function

Sets the dynamic class name.

#### Syntax

```
public void SetExceptionType(const System.String<char>& exceptionType_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| exceptionType_ | const System.String<char>& | A dynamic class name. |

### SetFile(const System.String<char>&) Member Function

Sets the source file.

#### Syntax

```
public void SetFile(const System.String<char>& file_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| file_ | const System.String<char>& | A path to the source file. |

### SetLine(int) Member Function

Sets the source line number.

#### Syntax

```
public void SetLine(int line_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| line_ | int | A source line number. |

### ToString() const Member Function

Returns the error message with the full name of the thrown exception class, and source file path and source line number where the exception was thrown.

### Syntax

```
public System.String<char> ToString() const;
```

### Returns

System.String<char>

Returns the error message with the full name of the thrown exception class, and source file path and source line number where the exception was thrown.

### ~Exception() Member Function

Destructor.

#### Syntax

```
public ~Exception();
```

## 2.2.20 ExceptionPtr Class

The **ExceptionPtr** class can be used to hold an exception with full type information.

### Syntax

```
public class ExceptionPtr;
```

#### 2.2.20.1 Remarks

The **ExceptionPtr** is used with functions CaptureCurrentException() and RethrowException(System.ExceptionPtr&).

#### 2.2.20.2 Example

```
using System;

// This program writes the following message to the standard error stream:
// FooException at 'C:/Temp/exptr/System.ExceptionPtr.cm' line 32:
// exception from thread
// call stack:
// 1> function 'foo()' file C:/Temp/exptr/System.ExceptionPtr.cm line 32
// 0> function 'ThreadFunction(void*)' file C:/Temp/exptr/System.
    ExceptionPtr.cm line 40

public class FooException : Exception
{
    public FooException(const string& message_) : base(message_)
    {
    }
}

public class ThreadData
{
    public void SetException(ExceptionPtr&& exPtr_)
    {
        exPtr = exPtr_;
    }
    public ExceptionPtr GetException() const
    {
        return Rvalue(exPtr);
    }
    private ExceptionPtr exPtr;
}

void foo()
{
    throw FooException("exception from thread");
}

void ThreadFunction(void* data)
{
    ThreadData* threadData = cast<ThreadData*>(data);
    try
    {
        foo();
    }
```

```
    catch (const Exception& ex)
    {
        ExceptionPtr exPtr = CaptureCurrentException();
        threadData->SetException(Rvalue(exPtr));
    }
}

void main()
{
    try
    {
        ThreadData threadData;
        System.Threading.Thread thread(System.Threading.ThreadFun(
            ThreadFunction), &threadData);
        thread.Join();
        ExceptionPtr exPtr = threadData.GetException();
        if (exPtr.HasException())
        {
            RethrowException(exPtr);
        }
    }
    catch (const Exception& ex)
    {
        Console.Error() << ex.ToString() << endl();
    }
}
```

#### 2.2.20.3 Member Functions

| Member Function | Description |
|---|---|
| ExceptionPtr() | Default constructor. Initializes the **ExceptionPtr** empty. |
| ExceptionPtr(System.ExceptionPtr&&) | Move constructor. |
| operator=(System.ExceptionPtr&&) | Move assignment. |
| ExceptionId() const | Returns identifier of the captured exception, or 0 if the **ExceptionPtr** does not hold an exception. |
| ExceptionPtr(int, System.Exception*) | Constructor. Initializes the **ExceptionPtr** to hold an exception with the given identifier and exception. |
| HasException() const | Returns true if the **ExceptionPtr** holds an exception, false otherwise. |
| Release() const | Releases the captured exception and sets the **ExceptionPtr** empty. |

| | |
|---|---|
| ~ExceptionPtr() | Destroyes the exception if the **ExceptionPtr** has one. |

### ExceptionPtr() Member Function

Default constructor. Initializes the ExceptionPtr empty.

### Syntax

```
public ExceptionPtr();
```

### ExceptionPtr(System.ExceptionPtr&&) Member Function

Move constructor.

#### Syntax

```
public ExceptionPtr(System.ExceptionPtr&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ExceptionPtr&& | A ExceptionPtr to move from. |

### operator=(System.ExceptionPtr&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.ExceptionPtr&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ExceptionPtr&& | A ExceptionPtr to assign. |

### ExceptionId() const Member Function

Returns identifier of the captured exception, or 0 if the ExceptionPtr does not hold an exception.

#### Syntax

```
public int ExceptionId() const;
```

#### Returns

int

Returns identifier of the captured exception, or 0 if the ExceptionPtr does not hold an exception.

### ExceptionPtr(int, System.Exception*) Member Function

Constructor. Initializes the ExceptionPtr to hold an exception with the given identifier and exception.

#### Syntax

```
public ExceptionPtr(int exceptionId_, System.Exception* exception_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| exceptionId_ | int | Identifier of the captured exception. |
| exception_ | System.Exception* | Captured exception. |

### HasException() const Member Function

Returns true if the ExceptionPtr holds an exception, false otherwise.

#### Syntax

```
public bool HasException() const;
```

#### Returns

bool

Returns true if the ExceptionPtr holds an exception, false otherwise.

### Release() const Member Function

Releases the captured exception and sets the ExceptionPtr empty.

### Syntax

```
public System.Exception* Release() const;
```

### Returns

System.Exception*

Returns captured exception.

### ~ExceptionPtr() Member Function

Destroyes the exception if the ExceptionPtr has one.

#### Syntax

```
public ~ExceptionPtr();
```

## 2.2.21 FrontInsertIterator<C> Class

An output iterator that inserts elements to the front of a container.

### Syntax

```
public class FrontInsertIterator<C>;
```

### Constraint

C is FrontInsertionSequence

### Model of

OutputIterator<T>

#### 2.2.21.1 Remarks

System.FrontInserter.C.C.is.FrontInsertionSequence.C.ref is a helper function that returns a **FrontInsertIterator<C>** for a container.

#### 2.2.21.2 Example

```
using System;
using System.Collections;

//  Writes:
//  list: 3, 2, 1

void main()
{
    Set<int> s;
    s.Insert(2);
    s.Insert(1);
    s.Insert(3);
//  Set is a sorted container, so it contains now 1, 2, 3..

    List<int> list; // list is a model of a front insertion sequence

//  Copy each element in the set to the front of list using
    FrontInsertIterator...
    Copy(s.CBegin(), s.CEnd(), FrontInserter(list));

//  ForwardContainers containing ints can be put to OutputStream...
    Console.Out() << "list: " << list << endl();
}
```

#### 2.2.21.3 Type Definitions

| Name | Type | Description |
|---|---|---|
| PointerType | System.FrontInsertProxy<C>* | Pointer to implementation defined proxy type. |
| ReferenceType | System.FrontInsertProxy<C>& | Reference to implementation defined proxy type. |
| ValueType | System.FrontInsertProxy<C> | Implementation defined proxy type. |

#### 2.2.21.4 Member Functions

| Member Function | Description |
|---|---|
| FrontInsertIterator<C>() | Constructor. Default constructs a front insert iterator. |
| FrontInsertIterator<C>(const System.-FrontInsertIterator<C>&) | Copy constructor. |
| operator=(const System.FrontInsertIterator<-C>&) | Copy assignment. |
| FrontInsertIterator<C>(System.-FrontInsertIterator<C>&&) | Move constructor. |
| operator=(System.FrontInsertIterator<C>&&) | Move assignment. |
| FrontInsertIterator<C>(C&) | Copy constructor. |
| operator*() | Returns a reference to a proxy object that inserts values at the beginning of a container. |
| operator++() | Advances the iterator to the next element. |
| operator->() | Returns a pointer to a proxy object that inserts values at the beginning of a container. |

### FrontInsertIterator<C>() Member Function

Constructor. Default constructs a front insert iterator.

#### Syntax

```
public FrontInsertIterator<C>();
```

### FrontInsertIterator<C>(const System.FrontInsertIterator<C>&) Member Function

Copy constructor.

#### Syntax

```
public FrontInsertIterator<C>(const System.FrontInsertIterator<C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.FrontInsertIterator<C>& | Argument to copy. |

### operator=(const System.FrontInsertIterator<C>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.FrontInsertIterator<C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.FrontInsertIterator<C>& | Argument to assign. |

### FrontInsertIterator<C>(System.FrontInsertIterator<C>&&) Member Function

Move constructor.

#### Syntax

```
public FrontInsertIterator<C>(System.FrontInsertIterator<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.FrontInsertIterator<C>&& | Argument to move from. |

### operator=(System.FrontInsertIterator<C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.FrontInsertIterator<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.FrontInsertIterator<C>&& | Argument to assign from. |

### FrontInsertIterator<C>(C&) Member Function

Copy constructor.

#### Syntax

```
public FrontInsertIterator<C>(C& c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | A front insert iterator to copy. |

### operator*() Member Function

Returns a reference to a proxy object that inserts values at the beginning of a container.

### Syntax

```
public System.FrontInsertProxy<C>& operator*();
```

### Returns

System.FrontInsertProxy<C>&

Returns a reference to a proxy object that inserts values at the beginning of a container.

### operator++() Member Function

Advances the iterator to the next element.

#### Syntax

```
public System.FrontInsertIterator<C>& operator++();
```

#### Returns

System.FrontInsertIterator<C>&

Returns a reference to the iterator.

### operator->() Member Function

Returns a pointer to a proxy object that inserts values at the beginning of a container.

### Syntax

```
public System.FrontInsertProxy<C>* operator->();
```

### Returns

System.FrontInsertProxy<C>*

Returns a pointer to a proxy object that inserts values at the beginning of a container.

## 2.2.22 FrontInsertProxy<C> Class

Implementation detail.

### Syntax

```
public class FrontInsertProxy<C>;
```

### Constraint

C is FrontInsertionSequence

#### 2.2.22.1 Member Functions

| Member Function | Description |
|---|---|
| FrontInsertProxy<C>(System.-FrontInsertProxy<C>&&) | Move constructor. |
| operator=(System.FrontInsertProxy<C>&&) | Move assignment. |

### FrontInsertProxy<C>(System.FrontInsertProxy<C>&&) Member Function

Move constructor.

#### Syntax

```
public FrontInsertProxy<C>(System.FrontInsertProxy<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.FrontInsertProxy<C>&& | Argument to move from. |

### operator=(System.FrontInsertProxy<C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.FrontInsertProxy<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.FrontInsertProxy<C>&& | Argument to assign from. |

## 2.2.23 Greater2<T, U> Class

A *greater than* binary predicate.

### Syntax

```
public class Greater2<T, U>;
```

### Constraint

LessThanComparable<T, U>

### Model of

Relation<T, U, V>

### Base Class

System.BinaryPred<T, U>

#### 2.2.23.1 Remarks

T and U are possibly different types, but can be compared for less than relationship.

#### 2.2.23.2 Member Functions

| Member Function | Description |
|---|---|
| Greater2<T, U>() | Default constructor. |
| Greater2<T, U>(const System.Greater2<T, U>-&) | Copy constructor. |
| operator=(const System.Greater2<T, U>&) | Copy assignment. |
| Greater2<T, U>(System.Greater2<T, U>&&) | Move constructor. |
| operator=(System.Greater2<T, U>&&) | Move assignment. |
| operator()(const T&, const U&) const | Returns true if the first argument is greater than the second argument, false otherwise. |

### Greater2<T, U>() Member Function

Default constructor.

#### Syntax

```
public Greater2<T, U>();
```

### Greater2<T, U>(const System.Greater2<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public Greater2<T, U>(const System.Greater2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Greater2<T, U>& | Argument to copy. |

### operator=(const System.Greater2<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Greater2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Greater2<T, U>& | Argument to assign. |

### Greater2<T, U>(System.Greater2<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public Greater2<T, U>(System.Greater2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Greater2<T, U>&& | Argument to move from. |

### operator=(System.Greater2<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Greater2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Greater2<T, U>&& | Argument to assign from. |

### operator()(const T&, const U&) const Member Function

Returns true if the first argument is greater than the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const U& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const U& | The second argument. |

#### Returns

bool

Returns $left > right$.

### 2.2.24 Greater<T> Class

A *greater than* relation.

#### Syntax

```
public class Greater<T>;
```

#### Constraint

T is LessThanComparable

#### Model of

Relation<T>

#### Base Class

System.Rel<T>

#### 2.2.24.1 Example

```
using System;
using System.Collections;

//  Writes:
//  3, 2, 1

void main()
{
    List<int> list;
    list.Add(1);
    list.Add(2);
    list.Add(3);
    Sort(list, Greater<int>());
    Console.Out() << list << endl();
}
```

#### 2.2.24.2 Example

```
using System;
using System.Collections;
using System.IO;

//  Writes:
//  foo, baz, bar

class A
{
    public A(): id()
    {
    }
    public A(const string& id_): id(id_)
    {
```

```
    }
    public nothrow inline const string& Id() const
    {
        return id;
    }
    private string id;
}

// Note: need only to provide less than operator for A -- compiler
    implements >, <= and >=.

public nothrow inline bool operator<(const A& left, const A& right)
{
    return left.Id() < right.Id();
}

public OutputStream& operator<<(OutputStream& s, const List<A>& list)
{
    bool first = true;
    for (const A& a : list)
    {
        if (first)
        {
            first = false;
        }
        else
        {
            s.Write(", ");
        }
        s.Write(a.Id());
    }
    return s;
}

void main()
{
    List<A> list;
    A foo("foo");
    list.Add(foo);
    A bar("bar");
    list.Add(bar);
    A baz("baz");
    list.Add(baz);
    Sort(list, Greater<A>());
    Console.Out() << list << endl();
}
```

#### 2.2.24.3 Member Functions

| Member Function | Description |
|---|---|
| Greater<T>() | Default constructor. |
| Greater<T>(const System.Greater<T>&) | Copy constructor. |

| | |
|---|---|
| operator=(const System.Greater<T>&) | Copy assignment. |
| Greater<T>(System.Greater<T>&&) | Move constructor. |
| operator=(System.Greater<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns true if the first argument is greater than the second argument, false otherwise. |

### Greater<T>() Member Function

Default constructor.

### Syntax

```
public Greater<T>();
```

### Greater<T>(const System.Greater<T>&) Member Function

Copy constructor.

#### Syntax

```
public Greater<T>(const System.Greater<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Greater<T>& | Argument to copy. |

### operator=(const System.Greater<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Greater<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Greater<T>& | Argument to assign. |

### Greater<T>(System.Greater<T>&&) Member Function

Move constructor.

#### Syntax

```
public Greater<T>(System.Greater<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Greater<T>&& | Argument to move from. |

### operator=(System.Greater<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Greater<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Greater<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns true if the first argument is greater than the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const T& | The second argument. |

#### Returns

bool

Returns $left > right$.

## 2.2.25 GreaterOrEqualTo2<T, U> Class

A *greater than or equal to* binary predicate.

### Syntax

```
public class GreaterOrEqualTo2<T, U>;
```

### Constraint

LessThanComparable<T, U>

### Model of

Relation<T, U, V>

### Base Class

System.BinaryPred<T, U>

#### 2.2.25.1 Remarks

T and U are possible different types, but can be compared for less than relationship.

#### 2.2.25.2 Member Functions

| Member Function | Description |
|---|---|
| GreaterOrEqualTo2<T, U>() | Default constructor. |
| GreaterOrEqualTo2<T, U>(const System.-GreaterOrEqualTo2<T, U>&) | Copy constructor. |
| operator=(const System.GreaterOrEqualTo2<T, U>&) | Copy assignment. |
| GreaterOrEqualTo2<T, U>(System.-GreaterOrEqualTo2<T, U>&&) | Move constructor. |
| operator=(System.GreaterOrEqualTo2<T, U>-&&) | Move assignment. |
| operator()(const T&, const U&) const | Returns true if the first argument is greater than or equal to the second argument, false otherwise. |

### GreaterOrEqualTo2<T, U>() Member Function

Default constructor.

#### Syntax

```
public GreaterOrEqualTo2<T, U>();
```

### GreaterOrEqualTo2<T, U>(const System.GreaterOrEqualTo2<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public GreaterOrEqualTo2<T, U>(const System.GreaterOrEqualTo2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.GreaterOrEqualTo2<T, U>& | Argument to copy. |

### operator=(const System.GreaterOrEqualTo2<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.GreaterOrEqualTo2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.GreaterOrEqualTo2<T, U>& | Argument to assign. |

### GreaterOrEqualTo2<T, U>(System.GreaterOrEqualTo2<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public GreaterOrEqualTo2<T, U>(System.GreaterOrEqualTo2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.GreaterOrEqualTo2<T, U>&& | Argument to move from. |

### operator=(System.GreaterOrEqualTo2<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.GreaterOrEqualTo2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.GreaterOrEqualTo2<T, U>&& | Argument to assign from. |

### operator()(const T&, const U&) const Member Function

Returns true if the first argument is greater than or equal to the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const U& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const U& | The second argument. |

#### Returns

bool

Returns $left >= right$.

## 2.2.26 GreaterOrEqualTo<T> Class

A *greater than or equal to* relation.

### Syntax

```
public class GreaterOrEqualTo<T>;
```

### Constraint

T is LessThanComparable

### Model of

Relation<T>

### Base Class

System.Rel<T>

#### 2.2.26.1 Member Functions

| Member Function | Description |
|---|---|
| GreaterOrEqualTo<T>() | Default constructor. |
| GreaterOrEqualTo<T>(const System.-GreaterOrEqualTo<T>&) | Copy constructor. |
| operator=(const System.GreaterOrEqualTo<-T>&) | Copy assignment. |
| GreaterOrEqualTo<T>(System.-GreaterOrEqualTo<T>&&) | Move constructor. |
| operator=(System.GreaterOrEqualTo<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns true if the first argument is greater than or equal to the second argument, false otherwise. |

### GreaterOrEqualTo<T>() Member Function

Default constructor.

#### Syntax

```
public GreaterOrEqualTo<T>();
```

### GreaterOrEqualTo<T>(const System.GreaterOrEqualTo<T>&) Member Function

Copy constructor.

#### Syntax

```
public GreaterOrEqualTo<T>(const System.GreaterOrEqualTo<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.GreaterOrEqualTo<T>& | Argument to copy. |

### operator=(const System.GreaterOrEqualTo<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.GreaterOrEqualTo<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.GreaterOrEqualTo<T>& | Argument to assign. |

### GreaterOrEqualTo<T>(System.GreaterOrEqualTo<T>&&) Member Function

Move constructor.

#### Syntax

```
public GreaterOrEqualTo<T>(System.GreaterOrEqualTo<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.GreaterOrEqualTo<T>&& | Argument to move from. |

### operator=(System.GreaterOrEqualTo<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.GreaterOrEqualTo<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.GreaterOrEqualTo<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns true if the first argument is greater than or equal to the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const T& | The second argument. |

#### Returns

bool

Returns $left >= right$.

## 2.2.27 Handle Class

A RAII class for storing a file descriptor.

### Syntax

```
public class Handle;
```

#### 2.2.27.1 Member Functions

| Member Function | Description |
|---|---|
| Handle() | Default constructor. Initializes the file descriptor to -1. |
| Handle(System.Handle&&) | Move constructor. |
| operator=(System.Handle&&) | Move assignment. |
| operator_int() const | Conversion to integer. |
| ~Handle() | Destructor. If the contained file descriptor is not -1, closes the file by calling close() with the contained file descriptor. |

### Handle() Member Function

Default constructor. Initializes the file descriptor to -1.

### Syntax

```
public Handle();
```

### Handle(System.Handle&&) Member Function

Move constructor.

#### Syntax

```
public Handle(System.Handle&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Handle&& | A handle to move. |

### operator=(System.Handle&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Handle&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Handle&& | A handle to move. |

### operator_int() const Member Function

Conversion to integer.

#### Syntax

```
public int operator_int() const;
```

#### Returns

int

Returns the contained file descriptor value.

### ~Handle() Member Function

Destructor. If the contained file descriptor is not -1, closes the file by calling close() with the contained file descriptor.

#### Syntax

```
public ~Handle();
```

## 2.2.28 Identity<T> Class

An identity unary function object.

### Syntax

```
public class Identity<T>;
```

### Constraint

T is Semiregular

### Model of

UnaryOperation<T>

### Base Class

System.UnaryFun<T, T>

#### 2.2.28.1 Member Functions

| Member Function | Description |
|---|---|
| Identity<T>() | Default constructor. |
| Identity<T>(const System.Identity<T>&) | Copy constructor. |
| operator=(const System.Identity<T>&) | Copy assignment. |
| Identity<T>(System.Identity<T>&&) | Move constructor. |
| operator=(System.Identity<T>&&) | Move assignment. |
| operator()(const T&) const | Returns the argument. |

### Identity<T>() Member Function

Default constructor.

### Syntax

```
public Identity<T>();
```

### Identity<T>(const System.Identity<T>&) Member Function

Copy constructor.

#### Syntax

```
public Identity<T>(const System.Identity<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Identity<T>& | Argument to copy. |

### operator=(const System.Identity<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Identity<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Identity<T>& | Argument to assign. |

### Identity<T>(System.Identity<T>&&) Member Function

Move constructor.

#### Syntax

```
public Identity<T>(System.Identity<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Identity<T>&& | Argument to move from. |

### operator=(System.Identity<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Identity<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Identity<T>&& | Argument to assign from. |

### operator()(const T&) const Member Function

Returns the argument.

#### Syntax

```
public const T& operator()(const T& x) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | const T& | An argument. |

#### Returns

const T&

Returns $x$.

## 2.2.29 InsertIterator<C> Class

An output iterator that inserts elements to given position of a container.

### Syntax

```
public class InsertIterator<C>;
```

### Constraint

C is InsertionSequence

### Model of

OutputIterator<T>

#### 2.2.29.1 Remarks

System.Inserter.C.I.C.is.InsertionSequence.I.is.C.Iterator.C.ref.I is a helper function that returns a **InsertIterator<C>** for a container and position.

#### 2.2.29.2 Example

```
using System;
using System.Collections;

//  Writes:
//  0, 1, 2, 3, 4

void main()
{
    Set<int> s;
    s.Insert(3);
    s.Insert(2);
    s.Insert(1);
//  Set is a sorted container, so it contains now 1, 2, 3...

    List<int> list;
    list.Add(0);
    list.Add(4);

//  Copy the set in the middle of the list using InsertIterator...
    Copy(s.CBegin(), s.CEnd(), Inserter(list, list.Begin() + 1));

//  ForwardContainers containing ints can be put to OutputStream...
    Console.Out() << list << endl();
}
```

#### 2.2.29.3 Type Definitions

| Name | Type | Description |
|---|---|---|
| PointerType | System.InsertProxy<C>* | Pointer to an implementation defined proxy type. |
| ReferenceType | System.InsertProxy<C>& | Reference to an implementation defined proxy type. |
| ValueType | System.InsertProxy<C> | An implementation defined proxy type. |

#### 2.2.29.4 Member Functions

| Member Function | Description |
|---|---|
| InsertIterator<C>() | Constructor. Default constructs an insert iterator. |
| InsertIterator<C>(const System.InsertIterator<C>&) | Copy constructor. |
| operator=(const System.InsertIterator<C>&) | Copy assignment. |
| InsertIterator<C>(System.InsertIterator<C>&&) | Move constructor. |
| operator=(System.InsertIterator<C>&&) | Move assignment. |
| operator*() | Returns a reference to a proxy type that inserts values to the container. |
| operator++() | Advances the insert iterator to the next element. |
| operator->() | Returns a pointer to a proxy type that inserts values to the container. |

### InsertIterator<C>() Member Function

Constructor. Default constructs an insert iterator.

#### Syntax

```
public InsertIterator<C>();
```

### InsertIterator<C>(const System.InsertIterator<C>&) Member Function

Copy constructor.

#### Syntax

```
public InsertIterator<C>(const System.InsertIterator<C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.InsertIterator<C>& | Argument to copy. |

### operator=(const System.InsertIterator<C>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.InsertIterator<C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.InsertIterator<C>& | Argument to assign. |

### InsertIterator<C>(System.InsertIterator<C>&&) Member Function

Move constructor.

#### Syntax

```
public InsertIterator<C>(System.InsertIterator<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.InsertIterator<C>&& | Argument to move from. |

### operator=(System.InsertIterator<C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.InsertIterator<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.InsertIterator<C>&& | Argument to assign from. |

### operator*() Member Function

Returns a reference to a proxy type that inserts values to the container.

### Syntax

```
public System.InsertProxy<C>& operator*();
```

### Returns

System.InsertProxy<C>&

Returns a reference to a proxy type that inserts values to the container.

### operator++() Member Function

Advances the insert iterator to the next element.

#### Syntax

```
public System.InsertIterator<C>& operator++();
```

#### Returns

System.InsertIterator<C>&

Returns a reference to the iterator.

### operator->() Member Function

Returns a pointer to a proxy type that inserts values to the container.

### Syntax

```
public System.InsertProxy<C>* operator->();
```

### Returns

System.InsertProxy<C>*

Returns a pointer to a proxy type that inserts values to the container.

## 2.2.30 InsertProxy<C> Class

Implementation detail.

### Syntax

```
public class InsertProxy<C>;
```

### Constraint

C is InsertionSequence

#### 2.2.30.1 Member Functions

| Member Function | Description |
|---|---|
| InsertProxy<C>(System.InsertProxy<C>&&) | Move constructor. |
| operator=(System.InsertProxy<C>&&) | Move assignment. |

### InsertProxy<C>(System.InsertProxy<C>&&) Member Function

Move constructor.

#### Syntax

```
public InsertProxy<C>(System.InsertProxy<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.InsertProxy<C>&& | Argument to move from. |

### operator=(System.InsertProxy<C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.InsertProxy<C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.InsertProxy<C>&& | Argument to assign from. |

## 2.2.31 Less2<T, U> Class

A *less than* binary predicate.

### Syntax

```
public class Less2<T, U>;
```

### Constraint

LessThanComparable<T, U>

### Model of

Relation<T, U, V>

### Base Class

System.BinaryPred<T, U>

#### 2.2.31.1 Remarks

T and U are possibly different types, but can be compared for less than relationship.

#### 2.2.31.2 Member Functions

| Member Function | Description |
|---|---|
| Less2<T, U>() | Default constructor. |
| Less2<T, U>(const System.Less2<T, U>&) | Copy constructor. |
| operator=(const System.Less2<T, U>&) | Copy assignment. |
| Less2<T, U>(System.Less2<T, U>&&) | Move constructor. |
| operator=(System.Less2<T, U>&&) | Move assignment. |
| operator()(const T&, const U&) const | Returns true if the first argument is less than the second argument. |

### Less2<T, U>() Member Function

Default constructor.

### Syntax

```
public Less2<T, U>();
```

### Less2<T, U>(const System.Less2<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public Less2<T, U>(const System.Less2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Less2<T, U>& | Argument to copy. |

### operator=(const System.Less2<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Less2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Less2<T, U>& | Argument to assign. |

### Less2<T, U>(System.Less2<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public Less2<T, U>(System.Less2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Less2<T, U>&& | Argument to move from. |

### operator=(System.Less2<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Less2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Less2<T, U>&& | Argument to assign from. |

### operator()(const T&, const U&) const Member Function

Returns true if the first argument is less than the second argument.

#### Syntax

```
public bool operator()(const T& left, const U& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const U& | The second argument. |

#### Returns

bool

Returns $left < right$.

## 2.2.32 Less<T> Class

A *less than* relation.

### Syntax

```
public class Less<T>;
```

### Constraint

T is LessThanComparable

### Model of

Relation<T>

### Base Class

System.Rel<T>

#### 2.2.32.1 Example

```
using System;
using System.Collections;

//  Writes:
//  1, 2, 3

void main()
{
    List<int> list;
    list.Add(3);
    list.Add(2);
    list.Add(1);
    Sort(list, Less<int>());
    Console.Out() << list << endl();
}
```

#### 2.2.32.2 Example

```
using System;
using System.Collections;
using System.IO;

//  Writes:
//  bar, baz, foo

class A
{
    public A(): id()
    {
    }
    public A(const string& id_): id(id_)
    {
```

```
    }
    public nothrow inline const string& Id() const
    {
        return id;
    }
    private string id;
}

public nothrow inline bool operator<(const A& left, const A& right)
{
    return left.Id() < right.Id();
}

public OutputStream& operator<<(OutputStream& s, const List<A>& list)
{
    bool first = true;
    for (const A& a : list)
    {
        if (first)
        {
            first = false;
        }
        else
        {
            s.Write(", ");
        }
        s.Write(a.Id());
    }
    return s;
}

void main()
{
    List<A> list;
    A foo("foo");
    list.Add(foo);
    A bar("bar");
    list.Add(bar);
    A baz("baz");
    list.Add(baz);
    Sort(list, Less<A>());
    Console.Out() << list << endl();
}
```

### 2.2.32.3 Member Functions

| Member Function | Description |
|---|---|
| Less<T>() | Default constructor. |
| Less<T>(const System.Less<T>&) | Copy constructor. |
| operator=(const System.Less<T>&) | Copy assignment. |
| Less<T>(System.Less<T>&&) | Move constructor. |

| | |
|---|---|
| operator=(System.Less<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns true if the first argument is less than the second argument, false otherwise. |

### Less<T>() Member Function

Default constructor.

### Syntax

```
public Less<T>();
```

### Less<T>(const System.Less<T>&) Member Function

Copy constructor.

#### Syntax

```
public Less<T>(const System.Less<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Less<T>& | Argument to copy. |

### operator=(const System.Less<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Less<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Less<T>& | Argument to assign. |

### Less<T>(System.Less<T>&&) Member Function

Move constructor.

#### Syntax

```
public Less<T>(System.Less<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Less<T>&& | Argument to move from. |

### operator=(System.Less<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Less<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Less<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns true if the first argument is less than the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const T& | The second argument. |

#### Returns

bool

Returns $left < right$.

## 2.2.33 LessOrEqualTo2<T, U> Class

A *less than or equal to* binary predicate.

### Syntax

```
public class LessOrEqualTo2<T, U>;
```

### Constraint

LessThanComparable<T, U>

### Model of

Relation<T, U, V>

### Base Class

System.BinaryPred<T, U>

#### 2.2.33.1 Remarks

T and U are possibly different types, but can be compared for less than relationship.

#### 2.2.33.2 Member Functions

| Member Function | Description |
|---|---|
| LessOrEqualTo2<T, U>() | Default constructor. |
| LessOrEqualTo2<T, U>(const System.-LessOrEqualTo2<T, U>&) | Copy constructor. |
| operator=(const System.LessOrEqualTo2<T, U>&) | Copy assignment. |
| LessOrEqualTo2<T, U>(System.-LessOrEqualTo2<T, U>&&) | Move constructor. |
| operator=(System.LessOrEqualTo2<T, U>&&) | Move assignment. |
| operator()(const T&, const U&) const | Returns true if the first argument is less than or equal to the second argument, false otherwise. |

### LessOrEqualTo2<T, U>() Member Function

Default constructor.

#### Syntax

```
public LessOrEqualTo2<T, U>();
```

### LessOrEqualTo2<T, U>(const System.LessOrEqualTo2<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public LessOrEqualTo2<T, U>(const System.LessOrEqualTo2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LessOrEqualTo2<T, U>& | Argument to copy. |

### operator=(const System.LessOrEqualTo2<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.LessOrEqualTo2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LessOrEqualTo2<T, U>& | Argument to assign. |

### LessOrEqualTo2<T, U>(System.LessOrEqualTo2<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public LessOrEqualTo2<T, U>(System.LessOrEqualTo2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LessOrEqualTo2<T, U>&& | Argument to move from. |

### operator=(System.LessOrEqualTo2<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.LessOrEqualTo2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LessOrEqualTo2<T, U>&& | Argument to assign from. |

### operator()(const T&, const U&) const Member Function

Returns true if the first argument is less than or equal to the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const U& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const U& | The second argument. |

#### Returns

bool

Returns $left <= right$.

## 2.2.34 LessOrEqualTo<T> Class

A *less than or equal to* relation.

### Syntax

```
public class LessOrEqualTo<T>;
```

### Constraint

T is LessThanComparable

### Model of

Relation<T>

### Base Class

System.Rel<T>

#### 2.2.34.1 Member Functions

| Member Function | Description |
|---|---|
| LessOrEqualTo<T>() | Default constructor. |
| LessOrEqualTo<T>(const System.-LessOrEqualTo<T>&) | Copy constructor. |
| operator=(const System.LessOrEqualTo<T>&) | Copy assignment. |
| LessOrEqualTo<T>(System.LessOrEqualTo<-T>&&) | Move constructor. |
| operator=(System.LessOrEqualTo<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns true if the first argument is less than or equal to the second argument, false otherwise. |

### LessOrEqualTo<T>() Member Function

Default constructor.

#### Syntax

```
public LessOrEqualTo<T>();
```

### LessOrEqualTo<T>(const System.LessOrEqualTo<T>&) Member Function

Copy constructor.

#### Syntax

```
public LessOrEqualTo<T>(const System.LessOrEqualTo<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LessOrEqualTo<T>& | Argument to copy. |

### operator=(const System.LessOrEqualTo<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.LessOrEqualTo<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LessOrEqualTo<T>& | Argument to assign. |

### LessOrEqualTo<T>(System.LessOrEqualTo<T>&&) Member Function

Move constructor.

#### Syntax

```
public LessOrEqualTo<T>(System.LessOrEqualTo<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LessOrEqualTo<T>&& | Argument to move from. |

### operator=(System.LessOrEqualTo<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.LessOrEqualTo<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LessOrEqualTo<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns true if the first argument is less than or equal to the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const T& | The second argument. |

#### Returns

bool

Returns $left <= right$.

## 2.2.35 LogicalAnd<T> Class

A *logical AND* binary predicate.

### Syntax

```
public class LogicalAnd<T>;
```

### Default Template Arguments

T = bool

### Base Class

System.BinaryPred<bool, bool>

#### 2.2.35.1 Member Functions

| Member Function | Description |
|---|---|
| LogicalAnd<T>() | Default constructor. |
| LogicalAnd<T>(const System.LogicalAnd<T>-&) | Copy constructor. |
| operator=(const System.LogicalAnd<T>&) | Copy assignment. |
| LogicalAnd<T>(System.LogicalAnd<T>&&) | Move constructor. |
| operator=(System.LogicalAnd<T>&&) | Move assignment. |
| operator()(bool, bool) const | Returns true if both left and right arguments are true, false otherwise. |

### LogicalAnd<T>() Member Function

Default constructor.

#### Syntax

```
public LogicalAnd<T>();
```

### LogicalAnd<T>(const System.LogicalAnd<T>&) Member Function

Copy constructor.

#### Syntax

```
public LogicalAnd<T>(const System.LogicalAnd<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LogicalAnd<T>& | Argument to copy. |

### operator=(const System.LogicalAnd<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.LogicalAnd<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LogicalAnd<T>& | Argument to assign. |

### LogicalAnd<T>(System.LogicalAnd<T>&&) Member Function

Move constructor.

#### Syntax

```
public LogicalAnd<T>(System.LogicalAnd<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LogicalAnd<T>&& | Argument to move from. |

### operator=(System.LogicalAnd<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.LogicalAnd<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LogicalAnd<T>&& | Argument to assign from. |

### operator()(bool, bool) const Member Function

Returns true if both left and right arguments are true, false otherwise.

#### Syntax

```
public bool operator()(bool left, bool right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | bool | Left argument. |
| right | bool | Right argument. |

#### Returns

bool

Returns true if both left and right arguments are true, false otherwise.

## 2.2.36 LogicalNot<T> Class

A *logical NOT* unary predicate.

### Syntax

```
public class LogicalNot<T>;
```

### Default Template Arguments

T = bool

### Base Class

System.UnaryPred<bool>

#### 2.2.36.1 Member Functions

| Member Function | Description |
|---|---|
| LogicalNot<T>() | Default constructor. |
| LogicalNot<T>(const System.LogicalNot<T>&-) | Copy constructor. |
| operator=(const System.LogicalNot<T>&) | Copy assignment. |
| LogicalNot<T>(System.LogicalNot<T>&&) | Move constructor. |
| operator=(System.LogicalNot<T>&&) | Move assignment. |
| operator()(bool) const | Returns true if operand is false, false otherwise. |

### LogicalNot<T>() Member Function

Default constructor.

#### Syntax

```
public LogicalNot<T>();
```

### LogicalNot<T>(const System.LogicalNot<T>&) Member Function

Copy constructor.

#### Syntax

```
public LogicalNot<T>(const System.LogicalNot<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LogicalNot<T>& | Argument to copy. |

### operator=(const System.LogicalNot<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.LogicalNot<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LogicalNot<T>& | Argument to assign. |

### LogicalNot<T>(System.LogicalNot<T>&&) Member Function

Move constructor.

#### Syntax

```
public LogicalNot<T>(System.LogicalNot<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LogicalNot<T>&& | Argument to move from. |

### operator=(System.LogicalNot<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.LogicalNot<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LogicalNot<T>&& | Argument to assign from. |

### operator()(bool) const Member Function

Returns true if operand is false, false otherwise.

#### Syntax

```
public bool operator()(bool operand) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| operand | bool | Operand. |

#### Returns

bool

Returns true if operand is false, false otherwise.

## 2.2.37 LogicalOr<T> Class

A *logical OR* binary predicate.

### Syntax

```
public class LogicalOr<T>;
```

### Default Template Arguments

T = bool

### Base Class

System.BinaryPred<bool, bool>

#### 2.2.37.1 Member Functions

| Member Function | Description |
|---|---|
| LogicalOr<T>() | Default constructor. |
| LogicalOr<T>(const System.LogicalOr<T>&) | Copy constructor. |
| operator=(const System.LogicalOr<T>&) | Copy assignment. |
| LogicalOr<T>(System.LogicalOr<T>&&) | Move constructor. |
| operator=(System.LogicalOr<T>&&) | Move assignment. |
| operator()(bool, bool) const | Returns true if left is true, or right is true, or both are true, false otherwise. |

### LogicalOr<T>() Member Function

Default constructor.

#### Syntax

```
public LogicalOr<T>();
```

### LogicalOr<T>(const System.LogicalOr<T>&) Member Function

Copy constructor.

#### Syntax

```
public LogicalOr<T>(const System.LogicalOr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LogicalOr<T>& | Argument to copy. |

### operator=(const System.LogicalOr<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.LogicalOr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.LogicalOr<T>& | Argument to assign. |

### LogicalOr<T>(System.LogicalOr<T>&&) Member Function

Move constructor.

#### Syntax

```
public LogicalOr<T>(System.LogicalOr<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LogicalOr<T>&& | Argument to move from. |

### operator=(System.LogicalOr<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.LogicalOr<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.LogicalOr<T>&& | Argument to assign from. |

### operator()(bool, bool) const Member Function

Returns true if left is true, or right is true, or both are true, false otherwise.

#### Syntax

```
public bool operator()(bool left, bool right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | bool | Left argument. |
| right | bool | Right argument. |

#### Returns

bool

Returns true if left is true, or right is true, or both are true, false otherwise.

## 2.2.38 MT Class

Mersenne Twister pseudorandom number generator.

### Syntax

```
public static class MT;
```

### 2.2.38.1 Remarks

Rand() function returns a pseudorandom number in range 0..MaxValue(uint) inclusive.

### 2.2.38.2 Member Functions

| Member Function | Description |
|---|---|
| GenRand() | Generates a pseudorandom number in range 0..MaxValue(uint) and returns it. |
| Init(uint) | Initializes the pseudorandom number generator with the given seed. |
| InitWithRandomSeed() | Initializes the pseudorandom number generator with a random seed returned by the system. |

### GenRand() Member Function

Generates a pseudorandom number in range 0..MaxValue(uint) and returns it.

### Syntax

```
public static uint GenRand();
```

### Returns

uint

Generates a pseudorandom number in range 0..MaxValue(uint) and returns it.

### Init(uint) Member Function

Initializes the pseudorandom number generator with the given seed.

#### Syntax

```
public static void Init(uint seed);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| seed | uint | A seed value. |

#### Remarks

In general the **Init(uint)** function need not be called because the static constructor initializes the generator with random seed returned by the system.

### InitWithRandomSeed() Member Function

Initializes the pseudorandom number generator with a random seed returned by the system.

#### Syntax

```
public static void InitWithRandomSeed();
```

## 2.2.39 Minus<T> Class

A subtraction binary function object.

### Syntax

```
public class Minus<T>;
```

### Constraint

T is AdditiveGroup

### Model of

BinaryFunction<T>

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.39.1 Member Functions

| Member Function | Description |
|---|---|
| Minus<T>() | Default constructor. |
| Minus<T>(const System.Minus<T>&) | Copy constructor. |
| operator=(const System.Minus<T>&) | Copy assignment. |
| Minus<T>(System.Minus<T>&&) | Move constructor. |
| operator=(System.Minus<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns the difference of the the first and second argument. |

### Minus<T>() Member Function

Default constructor.

### Syntax

```
public Minus<T>();
```

### Minus<T>(const System.Minus<T>&) Member Function

Copy constructor.

#### Syntax

```
public Minus<T>(const System.Minus<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Minus<T>& | Argument to copy. |

### operator=(const System.Minus<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Minus<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Minus<T>& | Argument to assign. |

### Minus<T>(System.Minus<T>&&) Member Function

Move constructor.

#### Syntax

```
public Minus<T>(System.Minus<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Minus<T>&& | Argument to move from. |

### operator=(System.Minus<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Minus<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Minus<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns the difference of the the first and second argument.

#### Syntax

```
public T operator()(const T& a, const T& b) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first argument. |
| b | const T& | The second argument. |

#### Returns

T

Returns $a - b$.

## 2.2.40 Multiplies<T> Class

A multiplication binary function object.

### Syntax

```
public class Multiplies<T>;
```

### Constraint

T is MultiplicativeSemigroup

### Model of

BinaryOperation<T>

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.40.1 Example

```
using System;
using System.Collections;

//  Writes:
//  48

void main()
{
    List<int> ints;
    ints.Add(2);
    ints.Add(4);
    ints.Add(6);
    int init = 1;
    int product = Accumulate(ints.CBegin(), ints.CEnd(), init, Multiplies<
        int>());
    Console.WriteLine(product);
}
```

#### 2.2.40.2 Member Functions

| Member Function | Description |
|---|---|
| Multiplies<T>() | Default constructor. |
| Multiplies<T>(const System.Multiplies<T>&) | Copy constructor. |
| operator=(const System.Multiplies<T>&) | Copy assignment. |
| Multiplies<T>(System.Multiplies<T>&&) | Move constructor. |
| operator=(System.Multiplies<T>&&) | Move assignment. |

| | |
|---|---|
| operator()(const T&, const T&) const | Returns the first argument multiplied with the second argument. |

### Multiplies<T>() Member Function

Default constructor.

### Syntax

```
public Multiplies<T>();
```

### Multiplies<T>(const System.Multiplies<T>&) Member Function

Copy constructor.

#### Syntax

```
public Multiplies<T>(const System.Multiplies<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Multiplies<T>& | Argument to copy. |

### operator=(const System.Multiplies<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Multiplies<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Multiplies<T>& | Argument to assign. |

### Multiplies<T>(System.Multiplies<T>&&) Member Function

Move constructor.

#### Syntax

```
public Multiplies<T>(System.Multiplies<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Multiplies<T>&& | Argument to move from. |

### operator=(System.Multiplies<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Multiplies<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Multiplies<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns the first argument multiplied with the second argument.

#### Syntax

```
public T operator()(const T& a, const T& b) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first argument. |
| b | const T& | The second argument. |

#### Returns

T

Returns $a * b$.

## 2.2.41 Negate<T> Class

A negation unary operation.

### Syntax

```
public class Negate<T>;
```

### Constraint

T is AdditiveGroup

### Model of

UnaryOperation<T>

### Base Class

System.UnaryFun<T, T>

#### 2.2.41.1 Member Functions

| Member Function | Description |
|---|---|
| Negate<T>() | Default constructor. |
| Negate<T>(const System.Negate<T>&) | Copy constructor. |
| operator=(const System.Negate<T>&) | Copy assignment. |
| Negate<T>(System.Negate<T>&&) | Move constructor. |
| operator=(System.Negate<T>&&) | Move assignment. |
| operator()(const T&) const | Returns the negation of its argument. |

### Negate<T>() Member Function

Default constructor.

### Syntax

```
public Negate<T>();
```

### Negate<T>(const System.Negate<T>&) Member Function

Copy constructor.

#### Syntax

```
public Negate<T>(const System.Negate<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Negate<T>& | Argument to copy. |

### operator=(const System.Negate<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Negate<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Negate<T>& | Argument to assign. |

### Negate<T>(System.Negate<T>&&) Member Function

Move constructor.

#### Syntax

```
public Negate<T>(System.Negate<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Negate<T>&& | Argument to move from. |

### operator=(System.Negate<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Negate<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Negate<T>&& | Argument to assign from. |

### operator()(const T&) const Member Function

Returns the negation of its argument.

#### Syntax

```
public T operator()(const T& a) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The argument. |

#### Returns

T

Returns $-a$.

## 2.2.42 NotEqualTo2<T, U> Class

A *not equal to* binary predicate.

### Syntax

```
public class NotEqualTo2<T, U>;
```

### Constraint

EqualityComparable<T, U>

### Model of

Relation<T, U, V>

### Base Class

System.BinaryPred<T, U>

#### 2.2.42.1 Remarks

T and U are possibly different types, but can be compared for equality.

#### 2.2.42.2 Member Functions

| Member Function | Description |
|---|---|
| NotEqualTo2<T, U>() | Default constructor. |
| NotEqualTo2<T, U>(const System.-NotEqualTo2<T, U>&) | Copy constructor. |
| operator=(const System.NotEqualTo2<T, U>&) | Copy assignment. |
| NotEqualTo2<T, U>(System.NotEqualTo2<T, U>&&) | Move constructor. |
| operator=(System.NotEqualTo2<T, U>&&) | Move assignment. |
| operator()(const T&, const U&) const | Returns true if the first argument is not equal to the second argument, false otherwise. |

### NotEqualTo2<T, U>() Member Function

Default constructor.

#### Syntax

```
public NotEqualTo2<T, U>();
```

### NotEqualTo2<T, U>(const System.NotEqualTo2<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public NotEqualTo2<T, U>(const System.NotEqualTo2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.NotEqualTo2<T, U>& | Argument to copy. |

### operator=(const System.NotEqualTo2<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.NotEqualTo2<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.NotEqualTo2<T, U>& | Argument to assign. |

### NotEqualTo2<T, U>(System.NotEqualTo2<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public NotEqualTo2<T, U>(System.NotEqualTo2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.NotEqualTo2<T, U>&& | Argument to move from. |

### operator=(System.NotEqualTo2<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.NotEqualTo2<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.NotEqualTo2<T, U>&& | Argument to assign from. |

### operator()(const T&, const U&) const Member Function

Returns true if the first argument is not equal to the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const U& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const U& | The second argument. |

#### Returns

bool

Returns $left$ != $right$.

## 2.2.43 NotEqualTo<T> Class

An *not equal to* relation.

### Syntax

```
public class NotEqualTo<T>;
```

### Constraint

T is Regular

### Model of

Relation<T>

### Base Class

System.Rel<T>

#### 2.2.43.1 Member Functions

| Member Function | Description |
|---|---|
| NotEqualTo<T>() | Default constructor. |
| NotEqualTo<T>(const System.NotEqualTo<-T>&) | Copy constructor. |
| operator=(const System.NotEqualTo<T>&) | Copy assignment. |
| NotEqualTo<T>(System.NotEqualTo<T>&&) | Move constructor. |
| operator=(System.NotEqualTo<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns true if the first argument is not equal to the second argument, false otherwise. |

### NotEqualTo<T>() Member Function

Default constructor.

#### Syntax

```
public NotEqualTo<T>();
```

### NotEqualTo<T>(const System.NotEqualTo<T>&) Member Function

Copy constructor.

#### Syntax

```
public NotEqualTo<T>(const System.NotEqualTo<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.NotEqualTo<T>& | Argument to copy. |

### operator=(const System.NotEqualTo<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.NotEqualTo<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.NotEqualTo<T>& | Argument to assign. |

### NotEqualTo<T>(System.NotEqualTo<T>&&) Member Function

Move constructor.

#### Syntax

```
public NotEqualTo<T>(System.NotEqualTo<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.NotEqualTo<T>&& | Argument to move from. |

### operator=(System.NotEqualTo<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.NotEqualTo<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.NotEqualTo<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns true if the first argument is not equal to the second argument, false otherwise.

#### Syntax

```
public bool operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first argument. |
| right | const T& | The second argument. |

#### Returns

bool

Returns $left$ != $right$.

## 2.2.44 Pair<T, U> Class

A pair of values.

### Syntax

```
public class Pair<T, U>;
```

### Constraint

T is Semiregular and U is Semiregular

#### 2.2.44.1 Remarks

The **Pair<T, U>** class is used for example in the system library algorithm System.EqualRange.I.T.I.is.ForwardIterator.and.TotallyOrdered.T.I.ValueType.I.I.T.const.ref to return a pair of iterators, and in the system library Map<Key, Value, KeyCompare> class to compose a value type from a key type and a mapped type.

#### 2.2.44.2 Member Functions

| Member Function | Description |
|---|---|
| Pair<T, U>() | Constructor. Constructs a pair with default values. |
| Pair<T, U>(const System.Pair<T, U>&) | Copy constructor. |
| operator=(const System.Pair<T, U>&) | Copy assignment. |
| Pair<T, U>(System.Pair<T, U>&&) | Move constructor. |
| operator=(System.Pair<T, U>&&) | Move assignment. |
| Pair<T, U>(const T&, const U&) | Constructor. Constructs a pair with specified values. |

### Pair<T, U>() Member Function

Constructor. Constructs a pair with default values.

#### Syntax

```
public Pair<T, U>();
```

### Pair<T, U>(const System.Pair<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public Pair<T, U>(const System.Pair<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Pair<T, U>& | Argument to copy. |

### operator=(const System.Pair<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Pair<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Pair<T, U>& | Argument to assign. |

### Pair<T, U>(System.Pair<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public Pair<T, U>(System.Pair<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Pair<T, U>&& | Argument to move from. |

### operator=(System.Pair<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Pair<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Pair<T, U>&& | Argument to assign from. |

**Pair<T, U>(const T&, const U&) Member Function**

Constructor. Constructs a pair with specified values.

**Syntax**

```
public Pair<T, U>(const T& first_, const U& second_);
```

**Parameters**

| Name | Type | Description |
|---|---|---|
| first_ | const T& | The first value. |
| second_ | const U& | The second value. |

### 2.2.44.3 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.Pair<T, U>&, const System.Pair<T, U>&) | Compares two pairs for less than relationship. |
| operator==(const System.Pair<T, U>&, const System.Pair<T, U>&) | Compares two pairs for equality. |

### operator<(const System.Pair<T, U>&, const System.Pair<T, U>&) Function

Compares two pairs for less than relationship.

#### Syntax

```
public bool operator<(const System.Pair<T, U>& left, const System.Pair<T, U>& right);
```

## Constraint

T is TotallyOrdered and U is TotallyOrdered

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Pair<T, U>& | The first pair. |
| right | const System.Pair<T, U>& | The second pair. |

#### Returns

bool

Returns true if the first component of the first pair is less than the first component of the second pair. Returns false if the first component of the second pair is less than the first component of the first pair. Returns true if the second component of the first pair is less than the second component of the second pair. Otherwise returns false.

### operator==(const System.Pair<T, U>&, const System.Pair<T, U>&) Function

Compares two pairs for equality.

#### Syntax

```
public bool operator==(const System.Pair<T, U>& left, const System.Pair<T, U>& right);
```

## Constraint

T is Regular and U is Regular

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Pair<T, U>& | The first pair. |
| right | const System.Pair<T, U>& | The second pair. |

#### Returns

bool

Returns true if the first component of the first pair is equal to the first component of the second pair and the second component of the first pair is equal to the second component of the second pair, false otherwise.

## 2.2.45 Plus<T> Class

An addition binary function object.

### Syntax

```
public class Plus<T>;
```

### Constraint

T is AdditiveSemigroup

### Model of

BinaryOperation<T>

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.45.1 Example

```
using System;
using System.Collections;

//  Writes:
//  6

void main()
{
    List<int> ints;
    ints.Add(1);
    ints.Add(2);
    ints.Add(3);
    int init = 0;
    int sum = Accumulate(ints.CBegin(), ints.CEnd(), init, Plus<int>());
    Console.WriteLine(sum);
}
```

#### 2.2.45.2 Member Functions

| Member Function | Description |
|---|---|
| Plus<T>() | Default constructor. |
| Plus<T>(const System.Plus<T>&) | Copy constructor. |
| operator=(const System.Plus<T>&) | Copy assignment. |
| Plus<T>(System.Plus<T>&&) | Move constructor. |
| operator=(System.Plus<T>&&) | Move assignment. |

| | |
|---|---|
| operator()(const T&, const T&) const | Returns the sum of the first argument and the second argument. |

### Plus<T>() Member Function

Default constructor.

### Syntax

```
public Plus<T>();
```

### Plus<T>(const System.Plus<T>&) Member Function

Copy constructor.

#### Syntax

```
public Plus<T>(const System.Plus<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Plus<T>& | Argument to copy. |

### operator=(const System.Plus<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Plus<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Plus<T>& | Argument to assign. |

### Plus<T>(System.Plus<T>&&) Member Function

Move constructor.

#### Syntax

```
public Plus<T>(System.Plus<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Plus<T>&& | Argument to move from. |

### operator=(System.Plus<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Plus<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Plus<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns the sum of the first argument and the second argument.

#### Syntax

```
public T operator()(const T& a, const T& b) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | the first argument. |
| b | const T& | The second argument. |

#### Returns

T

Returns $a + b$.

## 2.2.46 RandomAccessIter<T, R, P> Class

A random access iterator that contains a pointer to elements.

### Syntax

```
public class RandomAccessIter<T, R, P>;
```

### Model of

RandomAccessIterator<T>

#### 2.2.46.1 Remarks

List<T> and System.String classes implement their iterator types using the **RandomAccessIter<T, R, P>** class.

### 2.2.46.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| PointerType | P | A pointer to an element. |
| ReferenceType | R | A reference to an element. |
| ValueType | T | The type of an element. |

### 2.2.46.3 Member Functions

| Member Function | Description |
|---|---|
| RandomAccessIter<T, R, P>() | Constructor. Default constructs a random access iterator. |
| RandomAccessIter<T, R, P>(const System.-RandomAccessIter<T, R, P>&) | Copy constructor. |
| operator=(const System.RandomAccessIter<T, R, P>&) | Copy assignment. |
| RandomAccessIter<T, R, P>(System.-RandomAccessIter<T, R, P>&&) | Move constructor. |
| operator=(System.RandomAccessIter<T, R, P>-&&) | Move assignment. |
| GetPtr() const | Returns the contained pointer. |
| RandomAccessIter<T, R, P>(P) | Constructor. Constructs a random access iterator with a pointer to elements. |
| operator*() const | Returns a reference to the element currently pointed to. |
| operator++() | Advances the random access iterator to point to the succeeding element. |
| operator–() | Backs the random access iterator to point to the preceding element. |
| operator->() const | Returns a pointer to the element currently pointed to. |
| operator[](int) const | Returns a reference to an element with a given index. |

### RandomAccessIter<T, R, P>() Member Function

Constructor. Default constructs a random access iterator.

#### Syntax

```
public RandomAccessIter<T, R, P>();
```

### RandomAccessIter<T, R, P>(const System.RandomAccessIter<T, R, P>&) Member Function

Copy constructor.

#### Syntax

```
public RandomAccessIter<T, R, P>(const System.RandomAccessIter<T, R, P>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.RandomAccessIter<T, R, P>& | Argument to copy. |

### operator=(const System.RandomAccessIter<T, R, P>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.RandomAccessIter<T, R, P>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.RandomAccessIter<T, R, P>& | Argument to assign. |

### RandomAccessIter<T, R, P>(System.RandomAccessIter<T, R, P>&&) Member Function

Move constructor.

#### Syntax

```
public RandomAccessIter<T, R, P>(System.RandomAccessIter<T, R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.RandomAccessIter<T, R, P>&& | Argument to move from. |

### operator=(System.RandomAccessIter<T, R, P>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.RandomAccessIter<T, R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.RandomAccessIter<T, R, P>&& | Argument to assign from. |

### GetPtr() const Member Function

Returns the contained pointer.

#### Syntax

```
public P GetPtr() const;
```

#### Returns

P

Returns *ptr*, where *ptr* is the contained pointer.

### RandomAccessIter<T, R, P>(P) Member Function

Constructor. Constructs a random access iterator with a pointer to elements.

#### Syntax

```
public RandomAccessIter<T, R, P>(P ptr_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | P | A pointer to elements. |

### operator*() const Member Function

Returns a reference to the element currently pointed to.

#### Syntax

```
public R operator*() const;
```

#### Returns

R

Returns $*ptr$, where $ptr$ is the contained pointer.

### operator++() Member Function

Advances the random access iterator to point to the succeeding element.

#### Syntax

```
public System.RandomAccessIter<T, R, P>& operator++();
```

#### Returns

System.RandomAccessIter<T, R, P>&

Returns a reference to the random access iterator.

### operator–() Member Function

Backs the random access iterator to point to the preceding element.

#### Syntax

```
public System.RandomAccessIter<T, R, P>& operator--();
```

#### Returns

System.RandomAccessIter<T, R, P>&

Returns a reference to the random access iterator.

### operator->() const Member Function

Returns a pointer to the element currently pointed to.

#### Syntax

```
public P operator->() const;
```

#### Returns

P

Returns *ptr*, where *ptr* is the contained pointer.

#### operator[](int) const Member Function

Returns a reference to an element with a given index.

#### Syntax

```
public R operator[](int index) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| index | int | An index. |

#### Returns

R

Returns *ptr*[*index*] where *ptr* is the contained pointer to elements.

### 2.2.46.4 Nonmember Functions

| Function | Description |
|---|---|
| operator+(const System.RandomAccessIter<T, R, P>&, int) | Returns a random access iterator advanced by the given integer offset. |
| operator+(int, const System.-RandomAccessIter<T, R, P>&) | Returns a random access iterator advanced by the given integer offset. |
| operator-(const System.RandomAccessIter<T, R, P>&, const System.RandomAccessIter<T, R, P>&) | Returns the distance between two random access iterators. |
| operator-(const System.RandomAccessIter<T, R, P>&, int) | Returns the difference of a random access iterator and an integer. |
| operator<(const System.RandomAccessIter<T, R, P>&, const System.RandomAccessIter<T, R, P>&) | Compares two random access iterators for less than relationship. |
| operator==(const System.RandomAccessIter<T, R, P>&, const System.RandomAccessIter<T, R, P>&) | Compares two random access iterators for equality. |

### operator+(const System.RandomAccessIter<T, R, P>&, int) Function

Returns a random access iterator advanced by the given integer offset.

#### Syntax

```
public System.RandomAccessIter<T, R, P> operator+(const System.RandomAccessIter<T,
R, P>& it, int offset);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| it | const System.RandomAccessIter<T, R, P>& | A random access iterator. |
| offset | int | An integer offset. |

#### Returns

System.RandomAccessIter<T, R, P>

Returns a random access iterator advanced by the given integer offset.

### operator+(int, const System.RandomAccessIter<T, R, P>&) Function

Returns a random access iterator advanced by the given integer offset.

#### Syntax

```
public System.RandomAccessIter<T, R, P> operator+(int offset, const System.RandomAccessIter<T,
R, P>& it);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| offset | int | An integer offset. |
| it | const System.RandomAccessIter<T, R, P>& | A random access iterator. |

#### Returns

System.RandomAccessIter<T, R, P>

Returns a random access iterator advanced by the given integer offset.

### operator-(const System.RandomAccessIter<T, R, P>&, const System.RandomAccessIter<T, R, P>&) Function

Returns the distance between two random access iterators.

#### Syntax

```
public int operator-(const System.RandomAccessIter<T, R, P>& left, const System.RandomAccessIter
R, P>& right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.RandomAccessIter<T, R, P>& | The first random access iterator. |
| right | const System.RandomAccessIter<T, R, P>& | The second random access iterator. |

#### Returns

int

Returns the difference of the pointer contained by *left* and the pointer contained by *right*.

#### Remarks

Both iterators must point to an element in the same sequence or both must be end iterators.

### operator-(const System.RandomAccessIter<T, R, P>&, int) Function

Returns the difference of a random access iterator and an integer.

#### Syntax

```
public System.RandomAccessIter<T, R, P> operator-(const System.RandomAccessIter<T,
R, P>& it, int offset);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| it | const System.RandomAccessIter<T, R, P>& | A random access iterator. |
| offset | int | An integer. |

#### Returns

System.RandomAccessIter<T, R, P>

Returns a random access iterator that comes *offset* elements before the *it*.

### operator<(const System.RandomAccessIter<T, R, P>&, const System.RandomAccessIter<T, R, P>&) Function

Compares two random access iterators for less than relationship.

#### Syntax

```
public bool operator<(const System.RandomAccessIter<T, R, P>& left, const System.RandomAccessIt
R, P>& right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.RandomAccessIter<T, R, P>& | The first random access iterator. |
| right | const System.RandomAccessIter<T, R, P>& | The second random access iterator. |

#### Returns

bool

Returns true if the element pointed by *left* comes before the element pointed by *right* in the sequence, false otherwise.

#### Remarks

Both iterators must point to an element of the same sequence or both must be end iterators.

### operator==(const System.RandomAccessIter<T, R, P>&, const System.RandomAccessIter<T, R, P>&) Function

Compares two random access iterators for equality.

#### Syntax

```
public bool operator==(const System.RandomAccessIter<T, R, P>& left, const System.RandomAccessIt
R, P>& right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.RandomAccessIter<T, R, P>& | The first random access iterator. |
| right | const System.RandomAccessIter<T, R, P>& | The second random access iterator. |

#### Returns

bool

Returns true if *left* points to the same element as *right* or both are end iterators, false otherwise.

## 2.2.47 Rel<Argument> Class

A base class for relation function objects.

### Syntax

```
public class Rel<Argument>;
```

### Constraint

Argument is Semiregular

### Model of

Relation<T>

### Base Class

System.BinaryPred<Argument, Argument>

#### 2.2.47.1 Remarks

EqualTo<T>, NotEqualTo<T>, Less<T>, Greater<T>, LessOrEqualTo<T> and GreaterOrEqualTo<T> classes derive from the **Rel<Argument>** class.

#### 2.2.47.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| Domain | Argument | The domain i.e. the type of the argument of the relation. |

#### 2.2.47.3 Member Functions

| Member Function | Description |
|---|---|
| Rel<Argument>() | Default constructor. |
| Rel<Argument>(const System.Rel<Argument>-&) | Copy constructor. |
| operator=(const System.Rel<Argument>&) | Copy assignment. |
| Rel<Argument>(System.Rel<Argument>&&) | Move constructor. |
| operator=(System.Rel<Argument>&&) | Move assignment. |

### Rel<Argument>() Member Function

Default constructor.

#### Syntax

```
public Rel<Argument>();
```

### Rel<Argument>(const System.Rel<Argument>&) Member Function

Copy constructor.

#### Syntax

```
public Rel<Argument>(const System.Rel<Argument>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Rel<Argument>& | Argument to copy. |

### operator=(const System.Rel<Argument>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Rel<Argument>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Rel<Argument>& | Argument to assign. |

### Rel<Argument>(System.Rel<Argument>&&) Member Function

Move constructor.

#### Syntax

```
public Rel<Argument>(System.Rel<Argument>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Rel<Argument>&& | Argument to move from. |

### operator=(System.Rel<Argument>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Rel<Argument>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Rel<Argument>&& | Argument to assign from. |

## 2.2.48 Remainder<T> Class

A remainder function object.

### Syntax

```
public class Remainder<T>;
```

### Constraint

T is EuclideanSemiring

### Model of

BinaryFunction<T>

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.48.1 Member Functions

| Member Function | Description |
|---|---|
| Remainder<T>() | Default constructor. |
| Remainder<T>(const System.Remainder<T>&) | Copy constructor. |
| operator=(const System.Remainder<T>&) | Copy assignment. |
| Remainder<T>(System.Remainder<T>&&) | Move constructor. |
| operator=(System.Remainder<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns the remainder of division of the first argument and the second argument. |

### Remainder<T>() Member Function

Default constructor.

#### Syntax

```
public Remainder<T>();
```

### Remainder<T>(const System.Remainder<T>&) Member Function

Copy constructor.

#### Syntax

```
public Remainder<T>(const System.Remainder<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Remainder<T>& | Argument to copy. |

### operator=(const System.Remainder<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Remainder<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Remainder<T>& | Argument to assign. |

### Remainder<T>(System.Remainder<T>&&) Member Function

Move constructor.

#### Syntax

```
public Remainder<T>(System.Remainder<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Remainder<T>&& | Argument to move from. |

### operator=(System.Remainder<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Remainder<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Remainder<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns the remainder of division of the first argument and the second argument.

#### Syntax

```
public T operator()(const T& a, const T& b) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first argument. |
| b | const T& | The second argument. |

#### Returns

T

Returns $a$ % $b$.

## 2.2.49 SelectFirst<T, U> Class

A function object for returning the first component of a pair.

### Syntax

```
public class SelectFirst<T, U>;
```

### Constraint

T is Semiregular and U is Semiregular

### Base Class

System.UnaryFun<System.Pair<T, U>, T>

#### 2.2.49.1 Member Functions

| Member Function | Description |
|---|---|
| SelectFirst<T, U>() | Default constructor. |
| SelectFirst<T, U>(const System.SelectFirst<T, U>&) | Copy constructor. |
| operator=(const System.SelectFirst<T, U>&) | Copy assignment. |
| SelectFirst<T, U>(System.SelectFirst<T, U>&&) | Move constructor. |
| operator=(System.SelectFirst<T, U>&&) | Move assignment. |
| operator()(const System.Pair<T, U>&) const | Returns the first component of the given pair. |

### SelectFirst<T, U>() Member Function

Default constructor.

#### Syntax

```
public SelectFirst<T, U>();
```

### SelectFirst<T, U>(const System.SelectFirst<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public SelectFirst<T, U>(const System.SelectFirst<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SelectFirst<T, U>& | Argument to copy. |

### operator=(const System.SelectFirst<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.SelectFirst<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SelectFirst<T, U>& | Argument to assign. |

### SelectFirst<T, U>(System.SelectFirst<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public SelectFirst<T, U>(System.SelectFirst<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.SelectFirst<T, U>&& | Argument to move from. |

### operator=(System.SelectFirst<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.SelectFirst<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.SelectFirst<T, U>&& | Argument to assign from. |

### operator()(const System.Pair<T, U>&) const Member Function

Returns the first component of the given pair.

#### Syntax

```
public const T& operator()(const System.Pair<T, U>& p) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| p | const System.Pair<T, U>& | A pair of values. |

#### Returns

const T&

Returns the first component of *p*.

## 2.2.50 SelectSecond<T, U> Class

A function object for returning the second component of a pair.

### Syntax

```
public class SelectSecond<T, U>;
```

### Constraint

T is Semiregular and U is Semiregular

### Base Class

System.UnaryFun<System.Pair<T, U>, U>

#### 2.2.50.1 Member Functions

| Member Function | Description |
|---|---|
| SelectSecond<T, U>() | Default constructor. |
| SelectSecond<T, U>(const System.-SelectSecond<T, U>&) | Copy constructor. |
| operator=(const System.SelectSecond<T, U>&) | Copy assignment. |
| SelectSecond<T, U>(System.SelectSecond<T, U>&&) | Move constructor. |
| operator=(System.SelectSecond<T, U>&&) | Move assignment. |
| operator()(const System.Pair<T, U>&) const | Returns the second component of the given pair. |

### SelectSecond<T, U>() Member Function

Default constructor.

#### Syntax

```
public SelectSecond<T, U>();
```

### SelectSecond<T, U>(const System.SelectSecond<T, U>&) Member Function

Copy constructor.

#### Syntax

```
public SelectSecond<T, U>(const System.SelectSecond<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SelectSecond<T, U>& | Argument to copy. |

### operator=(const System.SelectSecond<T, U>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.SelectSecond<T, U>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SelectSecond<T, U>& | Argument to assign. |

### SelectSecond<T, U>(System.SelectSecond<T, U>&&) Member Function

Move constructor.

#### Syntax

```
public SelectSecond<T, U>(System.SelectSecond<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.SelectSecond<T, U>&& | Argument to move from. |

### operator=(System.SelectSecond<T, U>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.SelectSecond<T, U>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.SelectSecond<T, U>&& | Argument to assign from. |

### operator()(const System.Pair<T, U>&) const Member Function

Returns the second component of the given pair.

#### Syntax

```
public const U& operator()(const System.Pair<T, U>& p) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| p | const System.Pair<T, U>& | A pair of values. |

#### Returns

const U&

Returns the second component of *p*.

## 2.2.51 ShareableFromThis<T> Class

A class that implements the "shared from this" idiom.

### Syntax

```
public class ShareableFromThis<T>;
```

### 2.2.51.1 Remarks

By deriving a class from **ShareableFromThis<T>**
with itself as the template argument, you can obtain a shared pointer to the class in its member functions (other that constructors) provided that there is a "living" SharedPtr<T> to the object.

### 2.2.51.2 Example

```
using System;

public class C: ShareableFromThis<C>
{
    public SharedPtr<C> mf()
    {
        return GetSharedFromThis();
    }
}

void main()
{
    SharedPtr<C> c(new C());
//  ...
    C* rawPtr = c.GetPtr();
//  ...
    SharedPtr<C> p = rawPtr->mf();
}
```

### 2.2.51.3 Member Functions

| Member Function | Description |
|---|---|
| ShareableFromThis<T>() | Default constructor. |
| ShareableFromThis<T>(const System.-ShareableFromThis<T>&) | Copy constructor. |
| operator=(const System.ShareableFromThis<-T>&) | Copy assignment. |
| ShareableFromThis<T>(System.-ShareableFromThis<T>&&) | Move constructor. |
| operator=(System.ShareableFromThis<T>&&) | Move assignment. |
| GetSharedFromThis() const | Returns a shared pointer to the class object. |
| GetWeakThis() | Returns the contained weak pointer to the class object. |

### ShareableFromThis<T>() Member Function

Default constructor.

#### Syntax

```
public ShareableFromThis<T>();
```

### ShareableFromThis<T>(const System.ShareableFromThis<T>&) Member Function

Copy constructor.

#### Syntax

```
public ShareableFromThis<T>(const System.ShareableFromThis<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.ShareableFromThis<T>& | Argument to copy. |

### operator=(const System.ShareableFromThis<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.ShareableFromThis<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.ShareableFromThis<T>& | Argument to assign. |

### ShareableFromThis<T>(System.ShareableFromThis<T>&&) Member Function

Move constructor.

#### Syntax

```
public ShareableFromThis<T>(System.ShareableFromThis<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ShareableFromThis<T>&& | Argument to move from. |

### operator=(System.ShareableFromThis<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.ShareableFromThis<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ShareableFromThis<T>&& | Argument to assign from. |

### GetSharedFromThis() const Member Function

Returns a shared pointer to the class object.

#### Syntax

```
public System.SharedPtr<T> GetSharedFromThis() const;
```

#### Returns

System.SharedPtr<T>

Returns a shared pointer to the class object.

### GetWeakThis() Member Function

Returns the contained weak pointer to the class object.

### Syntax

```
public System.WeakPtr<T>& GetWeakThis();
```

### Returns

System.WeakPtr<T>&

Returns the contained weak pointer to the class object.

## 2.2.52 SharedCount<T> Class

A handle to a Counter<T> that maintains the use count portion of the counter.

### Syntax

```
public class SharedCount<T>;
```

#### 2.2.52.1 Member Functions

| Member Function | Description |
|---|---|
| SharedCount<T>() | Constructor. Initializes an empty shared count. |
| SharedCount<T>(const System.SharedCount<-T>&) | Constructor. Implementation detail. |
| operator=(const System.SharedCount<T>&) | Copy assignment. Decrements the use count of the old counter and increments the use count of the copied counter. |
| GetCounter() const | Returns the contained pointer to a counter. |
| GetUseCount() const | Returns the use count of the counter. |
| IsUnique() const | Returns true if there is exactly one SharedPtr<-T> to an object. |
| SharedCount<T>(System.Counter<T>*) | Copy constructor. Increments the use count. |
| SharedCount<T>(T*) | Constructor. Initializes the counter to contain a pointer to a counted object. |
| SharedCount<T>(const System.WeakCount<-T>&) | Constructor. Gets the counter from a WeakCount<T>. |
| Swap(System.SharedCount<T>&) | Exchanges the contents with another shared count. |
| ~SharedCount<T>() | Destructor. Decrements the use count. |

### SharedCount<T>() Member Function

Constructor. Initializes an empty shared count.

#### Syntax

```
public SharedCount<T>();
```

### SharedCount<T>(const System.SharedCount<T>&) Member Function

Constructor. Implementation detail.

#### Syntax

```
public SharedCount<T>(const System.SharedCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SharedCount<T>& | Pointer to the counter. |

### operator=(const System.SharedCount<T>&) Member Function

Copy assignment. Decrements the use count of the old counter and increments the use count of the copied counter.

#### Syntax

```
public void operator=(const System.SharedCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SharedCount<T>& | A shared count to assign. |

### GetCounter() const Member Function

Returns the contained pointer to a counter.

#### Syntax

```
public System.Counter<T>* GetCounter() const;
```

#### Returns

System.Counter<T>*

Returns the contained pointer to a counter.

### GetUseCount() const Member Function

Returns the use count of the counter.

#### Syntax

```
public int GetUseCount() const;
```

#### Returns

int

Returns the use count of the counter.

### IsUnique() const Member Function

Returns true if there is exactly one SharedPtr<T> to an object.

#### Syntax

```
public bool IsUnique() const;
```

#### Returns

bool

Returns true if there is exactly one SharedPtr<T> to an object.

### SharedCount<T>(System.Counter<T>*) Member Function

Copy constructor. Increments the use count.

#### Syntax

```
public SharedCount<T>(System.Counter<T>* counter_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| counter_ | System.Counter<T>* | A shared count to copy. |

### SharedCount<T>(T*) Member Function

Constructor. Initializes the counter to contain a pointer to a counted object.

#### Syntax

```
public SharedCount<T>(T* ptr_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to a counted object. |

### SharedCount<T>(const System.WeakCount<T>&) Member Function

Constructor. Gets the counter from a WeakCount<T>.

#### Syntax

```
public SharedCount<T>(const System.WeakCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.WeakCount<T>& | A weak count. |

### Swap(System.SharedCount<T>&) Member Function

Exchanges the contents with another shared count.

#### Syntax

```
public void Swap(System.SharedCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.SharedCount<T>& | A shared count to exchange contents with. |

#### ~SharedCount<T>() Member Function

Destructor. Decrements the use count.

**Syntax**

```
public ~SharedCount<T>();
```

### 2.2.52.2 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.SharedCount<T>&, const System.SharedCount<T>&) | Compares two shared count objects for less than relationship. |
| operator==(const System.SharedCount<T>&, const System.SharedCount<T>&) | Compares to shared count objects for equality. |

### operator<(const System.SharedCount<T>&, const System.SharedCount<T>&) Function

Compares two shared count objects for less than relationship.

#### Syntax

```
public bool operator<(const System.SharedCount<T>& left, const System.SharedCount<T>&
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.SharedCount<T>& | The first shared count. |
| right | const System.SharedCount<T>& | The second shared count. |

#### Returns

bool

Returns true if the memory address the counter contained by the *left* count is less than the memory address of the counter contained by the *right* count, false otherwise.

### operator==(const System.SharedCount<T>&, const System.SharedCount<T>&) Function

Compares to shared count objects for equality.

#### Syntax

```
public bool operator==(const System.SharedCount<T>& left, const System.SharedCount<T>&
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.SharedCount<T>& | The first shared count. |
| right | const System.SharedCount<T>& | The second shared count. |

#### Returns

bool

Returns true if *left* contains the same counter as *right* or both are empty, false otherwise.

### 2.2.53 SharedPtr<T> Class

A shared pointer to an object.

#### Syntax

```
public class SharedPtr<T>;
```

#### Model of

TotallyOrdered<T>

#### 2.2.53.1 Example

```
using System;
using System.Collections;
using Animals;

//  Writes:
//  Rose says meow
//  Rudolf says woof

namespace Animals
{
    public abstract class Animal
    {
        public Animal(const string& name_): name(name_)
        {
        }
        public virtual ~Animal()
        {
        }
        public const string& Name() const
        {
            return name;
        }
        public abstract void Talk();
        private string name;
    }

    public typedef SharedPtr<Animal> AnimalPtr;

    public class Dog: Animal
    {
        public Dog(const string& name_): base(name_)
        {
        }
        public override void Talk()
        {
            Console.Out() << Name() << " says woof" << endl();
        }
    }

    public class Cat: Animal
    {
        public Cat(const string& name_): base(name_)
        {
```

```
        }
        public override void Talk()
        {
            Console.Out() << Name() << " says meow" << endl();
        }
    }
}

void main()
{
    List<AnimalPtr> animals;
    animals.Add(AnimalPtr(new Cat("Rose")));
    animals.Add(AnimalPtr(new Dog("Rudolf")));
//  ...
    for (AnimalPtr animal : animals)
    {
        animal->Talk();
    }
}
```

### 2.2.53.2 Member Functions

| Member Function | Description |
|---|---|
| SharedPtr<T>() | Constructor. Constructs a null shared pointer. |
| SharedPtr<T>(const System.SharedPtr<T>&) | Copy constructor. Increments the use count. |
| operator=(const System.SharedPtr<T>&) | Copy assignment. Assigns another shared pointer. |
| GetCount() const | Returns the contained SharedCount<T>. |
| GetPtr() const | Returns the contained pointer to the counted object. |
| GetUseCount() const | Returns the use count. |
| IsNull() const | Returns true if the contained pointer is null, false otherwise. |
| IsUnique() const | Returns true if there is exactly one **SharedPtr<-T>** to an object. |
| Reset() | Resets the shared pointer to null. |
| Reset(T*) | Resets the shared pointer to contain a pointer to another counted object. |
| SharedPtr<T>(T*) | Constructor. Constructs a shared pointer to the given object. |

| | |
|---|---|
| SharedPtr<T>(T*, const System.-SharedCount<T>&) | Constructor. Implementation detail to support System.PtrCast.U.T.System.SharedPtr.T.-const.ref function. |
| SharedPtr<T>(const System.WeakPtr<T>&) | Constructor. Constructs a shared pointer from a weak pointer. |
| Swap(System.SharedPtr<T>&) | Exchanges the contents with another shared pointer. |
| operator*() const | Returns a reference to the counted object. |
| operator->() const | Returns the contained pointer to the counted object. |

### SharedPtr<T>() Member Function

Constructor. Constructs a null shared pointer.

#### Syntax

```
public SharedPtr<T>();
```

### SharedPtr<T>(const System.SharedPtr<T>&) Member Function

Copy constructor. Increments the use count.

#### Syntax

```
public SharedPtr<T>(const System.SharedPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SharedPtr<T>& | A shared pointer to copy. |

### operator=(const System.SharedPtr<T>&) Member Function

Copy assignment. Assigns another shared pointer.

#### Syntax

```
public void operator=(const System.SharedPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SharedPtr<T>& | A shared pointer to assign. |

### GetCount() const Member Function

Returns the contained SharedCount<T>.

#### Syntax

```
public const System.SharedCount<T>& GetCount() const;
```

#### Returns

const System.SharedCount<T>&

Returns the contained shared count.

### GetPtr() const Member Function

Returns the contained pointer to the counted object.

#### Syntax

```
public T* GetPtr() const;
```

#### Returns

T*

Returns the contained pointer to the counted object.

### GetUseCount() const Member Function

Returns the use count.

#### Syntax

```
public int GetUseCount() const;
```

#### Returns

int

Returns the use count.

### IsNull() const Member Function

Returns true if the contained pointer is null, false otherwise.

#### Syntax

```
public bool IsNull() const;
```

#### Returns

bool

Returns true if the contained pointer is null, false otherwise.

### IsUnique() const Member Function

Returns true if there is exactly one SharedPtr<T> to an object.

#### Syntax

```
public bool IsUnique() const;
```

#### Returns

bool

Returns true if there is exactly one SharedPtr<T> to an object.

### Reset() Member Function

Resets the shared pointer to null.

### Syntax

```
public void Reset();
```

### Reset(T*) Member Function

Resets the shared pointer to contain a pointer to another counted object.

#### Syntax

```
public void Reset(T* ptr_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to another counted object. |

### SharedPtr<T>(T*) Member Function

Constructor. Constructs a shared pointer to the given object.

#### Syntax

```
public SharedPtr<T>(T* ptr_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to an object. |

### SharedPtr<T>(T*, const System.SharedCount<T>&) Member Function

Constructor. Implementation detail to support
System.PtrCast.U.T.System.SharedPtr.T.const.ref function.

#### Syntax

```
public SharedPtr<T>(T* ptr_, const System.SharedCount<T>& count_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to counted object. |
| count_ | const System.SharedCount<T>& | A shared count. |

### SharedPtr<T>(const System.WeakPtr<T>&) Member Function

Constructor. Constructs a shared pointer from a weak pointer.

#### Syntax

```
public SharedPtr<T>(const System.WeakPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.WeakPtr<T>& | A weak pointer to an object. |

### Swap(System.SharedPtr<T>&) Member Function

Exchanges the contents with another shared pointer.

#### Syntax

```
public void Swap(System.SharedPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.SharedPtr<T>& | A shared pointer to exchange contents with. |

### operator*() const Member Function

Returns a reference to the counted object.

#### Syntax

```
public T& operator*() const;
```

#### Returns

T&

Returns a reference to the counted object.

**operator->() const Member Function**

Returns the contained pointer to the counted object.

**Syntax**

```
public T* operator->() const;
```

**Returns**

T*

Returns the contained pointer to the counted object.

### 2.2.53.3 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.SharedPtr<T>&, const System.SharedPtr<T>&) | Compares two shared pointers for less than relationship. |
| operator==(const System.SharedPtr<T>&, const System.SharedPtr<T>&) | Compares two shared pointers for equality. |

### operator<(const System.SharedPtr<T>&, const System.SharedPtr<T>&) Function

Compares two shared pointers for less than relationship.

#### Syntax

```
public bool operator<(const System.SharedPtr<T>& left, const System.SharedPtr<T>&
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.SharedPtr<T>& | The first shared pointer. |
| right | const System.SharedPtr<T>& | The second shared pointer. |

#### Returns

bool

Returns true if the memory address of the counted object pointed by *left* is less than the memory address of the counted object pointed by *right*, false otherwise.

### operator==(const System.SharedPtr<T>&, const System.SharedPtr<T>&) Function

Compares two shared pointers for equality.

#### Syntax

```
public bool operator==(const System.SharedPtr<T>& left, const System.SharedPtr<T>&
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.SharedPtr<T>& | The first shared pointer. |
| right | const System.SharedPtr<T>& | The second shared pointer. |

#### Returns

bool

Returns true if *left* points to the same object as *right* or both are null, false otherwise.

## 2.2.54 ShiftLeft<T> Class

Shift left binary function.

### Syntax

```
public class ShiftLeft<T>;
```

### Constraint

T is Semiregular

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.54.1 Member Functions

| Member Function | Description |
|---|---|
| ShiftLeft<T>() | Default constructor. |
| ShiftLeft<T>(const System.ShiftLeft<T>&) | Copy constructor. |
| operator=(const System.ShiftLeft<T>&) | Copy assignment. |
| ShiftLeft<T>(System.ShiftLeft<T>&&) | Move constructor. |
| operator=(System.ShiftLeft<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns left shifted left by right bit positions. |

### ShiftLeft<T>() Member Function

Default constructor.

#### Syntax

```
public ShiftLeft<T>();
```

### ShiftLeft<T>(const System.ShiftLeft<T>&) Member Function

Copy constructor.

#### Syntax

```
public ShiftLeft<T>(const System.ShiftLeft<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.ShiftLeft<T>& | Argument to copy. |

### operator=(const System.ShiftLeft<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.ShiftLeft<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.ShiftLeft<T>& | Argument to assign. |

### ShiftLeft<T>(System.ShiftLeft<T>&&) Member Function

Move constructor.

#### Syntax

```
public ShiftLeft<T>(System.ShiftLeft<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ShiftLeft<T>&& | Argument to move from. |

### operator=(System.ShiftLeft<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.ShiftLeft<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ShiftLeft<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns left shifted left by right bit positions.

#### Syntax

```
public T operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | Left argument. |
| right | const T& | Right argument. |

#### Returns

T

Returns left shifted left by right bit positions.

## 2.2.55 ShiftRight<T> Class

Shift right binary function.

### Syntax

```
public class ShiftRight<T>;
```

### Constraint

T is Semiregular

### Base Class

System.BinaryFun<T, T, T>

#### 2.2.55.1 Member Functions

| Member Function | Description |
|---|---|
| ShiftRight<T>() | Default constructor. |
| ShiftRight<T>(const System.ShiftRight<T>&) | Copy constructor. |
| operator=(const System.ShiftRight<T>&) | Copy assignment. |
| ShiftRight<T>(System.ShiftRight<T>&&) | Move constructor. |
| operator=(System.ShiftRight<T>&&) | Move assignment. |
| operator()(const T&, const T&) const | Returns left shifted right by right bit positions. |

### ShiftRight<T>() Member Function

Default constructor.

#### Syntax

```
public ShiftRight<T>();
```

### ShiftRight<T>(const System.ShiftRight<T>&) Member Function

Copy constructor.

#### Syntax

```
public ShiftRight<T>(const System.ShiftRight<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.ShiftRight<T>& | Argument to copy. |

### operator=(const System.ShiftRight<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.ShiftRight<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.ShiftRight<T>& | Argument to assign. |

### ShiftRight<T>(System.ShiftRight<T>&&) Member Function

Move constructor.

#### Syntax

```
public ShiftRight<T>(System.ShiftRight<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ShiftRight<T>&& | Argument to move from. |

### operator=(System.ShiftRight<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.ShiftRight<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.ShiftRight<T>&& | Argument to assign from. |

### operator()(const T&, const T&) const Member Function

Returns left shifted right by right bit positions.

#### Syntax

```
public T operator()(const T& left, const T& right) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | Left argument. |
| right | const T& | Right argument. |

#### Returns

T

Returns left shifted right by right bit positions.

### 2.2.56 String<CharT> Class

A string class template.

#### Syntax

```
public class String<CharT>;
```

### 2.2.56.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| ConstIterator | System.RandomAccessIter<CharT, const CharT&, const CharT*> | A constant iterator type. |
| Iterator | System.RandomAccessIter<CharT, CharT&, CharT*> | An iterator type. |

### 2.2.56.2 Member Functions

| Member Function | Description |
|---|---|
| String<CharT>() | Constructor. Initializes a string with the given NUL-terminated sequence of characters. |
| String<CharT>(const System.String<CharT>-&) | Copy constructor. |
| operator=(const System.String<CharT>&) | Copy assignment. |
| String<CharT>(System.String<CharT>&&) | Move constructor. |
| operator=(System.String<CharT>&&) | Move assignment. |
| Append(CharT) | Appends the given chacter to the end of this string. |
| Append(CharT, int) | Appends given number of occurrences of given chacter to the end of this string. |
| Append(const CharT*) | Appends a NUL-terminated sequence of characters to the end of the string. |
| Append(const CharT*, int) | Appends the given number of characters from a NUL-terminated sequence of characters to the end of this string. |
| Append(const System.String<CharT>&) | Appends the specified string to the end of this string. |
| Begin() | Returns an iterator to the beginning of the string. |
| Begin() const | Returns a constant iterator to the beginning of the string. |
| CBegin() const | Returns a constant iterator to the beginning of the string. |
| CEnd() const | Returns a constant iterator one past the end of the string. |

| | |
|---|---|
| Capacity() const | Returns the number of characters that the string can hold without allocating more memory for it. |
| Chars() const | Returns the string as a NUL-terminated sequence of characters. |
| Clear() | Makes the string empty. |
| End() | Returns an iterator to one past the end of the string. |
| End() const | Returns a constant iterator to one past the end of the string. |
| EndsWith(const System.String<CharT>&) const | Returns true, if the string ends with the specified substring, false otherwise. |
| Find(CharT) const | Returns the index of the first occurrence of the given character within this string, or -1 if the specified character does not occur in this string. |
| Find(CharT, int) const | Returns the index of the first occurrence of the given character within this string, or -1 if the specified character does not occur in this string. The search starts with the specified index. |
| Find(const System.String<CharT>&) const | Returns the starting index of the first occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string. |
| Find(const System.String<CharT>&, int) const | Returns the starting index of the first occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string. The search starts with the specified index. |
| IsEmpty() const | Returns true if the string is empty, false otherwise. |
| Length() const | Returns the length of the string. |
| RFind(CharT) const | Returns the index of the last occurrence of the given character within this string, or -1 if the specified character does not occur in this string. |
| RFind(CharT, int) const | Returns the index of the last occurrence of the given character within this string, or -1 if the specified character does not occur in this string. The search starts with the specified index. |

| | |
|---|---|
| RFind(const System.String<CharT>&) const | Returns the starting index of the last occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string. |
| RFind(const System.String<CharT>&, int) const | Returns the starting index of the last occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string. The search starts with the specified index. |
| Replace(CharT, CharT) | Replaces every occurrence of the given character with another character. |
| Reserve(int) | Reserves room for a string with the given number of characters. |
| Split(CharT) | Returns a list of substrings separated by the given character. |
| StartsWith(const System.String<CharT>&) const | Returns true if the string starts with the given substring, false otherwise. |
| String<CharT>(CharT) | Constructor. Constructs a string that has the given character. |
| String<CharT>(CharT, int) | Constructor. Constructs a string that has given number of the specified character. |
| String<CharT>(const CharT*) | Constructor. Constructs a string from a NUL-terminated sequence of characters. |
| String<CharT>(const CharT*, const CharT*) | Constructor. Constructs a string of characters included in range begin and end. |
| String<CharT>(const CharT*, int) | Constructor. Constructs a string from given number of characters of a NUL-terminated sequence of characters. |
| Substring(int) const | Returns a substring starting from the given index. |
| Substring(int, int) const | Returns a substring starting from the given index whose length is at most given number of characters. |
| operator<(const System.String<CharT>&) const | Compares a string for less than relationship with another string. |

| | |
|---|---|
| operator==(const System.String<CharT>&) const | Compares a string for equality with another string. |
| operator[](int) | Returns a reference to the character with the specified index. |
| operator[](int) const | Returns a character with the specified index. |
| ~String<CharT>() | Destructor. Releases the memory occupied by the string. |

### String<CharT>() Member Function

Constructor. Initializes a string with the given NUL-terminated sequence of characters.

#### Syntax

```
public String<CharT>();
```

### String<CharT>(const System.String<CharT>&) Member Function

Copy constructor.

#### Syntax

```
public String<CharT>(const System.String<CharT>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.String<CharT>& | A string to copy. |

### operator=(const System.String<CharT>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.String<CharT>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.String<CharT>& | A string to assign. |

### String<CharT>(System.String<CharT>&&) Member Function

Move constructor.

#### Syntax

```
public String<CharT>(System.String<CharT>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.String<CharT>&& | A string to move from. |

### operator=(System.String<CharT>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.String<CharT>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.String<CharT>&& | A string to move from. |

### Append(CharT) Member Function

Appends the given chacter to the end of this string.

#### Syntax

```
public System.String<CharT>& Append(CharT c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | CharT | A character to append. |

#### Returns

System.String<CharT>&

Returns a reference to this string.

### Append(CharT, int) Member Function

Appends given number of occurrences of given chacter to the end of this string.

#### Syntax

```
public System.String<CharT>& Append(CharT c, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | CharT | A character to append. |
| count | int | Number of times to append. |

#### Returns

System.String<CharT>&

Returns a reference to this string.

### Append(const CharT*) Member Function

Appends a NUL-terminated sequence of characters to the end of the string.

#### Syntax

```
public System.String<CharT>& Append(const CharT* that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const CharT* | A NUL-terminated sequence of characters to append. |

#### Returns

System.String<CharT>&

Returns a reference to this string.

### Append(const CharT*, int) Member Function

Appends the given number of characters from a NUL-terminated sequence of characters to the end of this string.

#### Syntax

```
public System.String<CharT>& Append(const CharT* that, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const CharT* | A NUL-terminated sequence of characters to append. |
| count | int | The maximum number of characters to append. |

#### Returns

System.String<CharT>&

Returns a reference to this string.

### Append(const System.String<CharT>&) Member Function

Appends the specified string to the end of this string.

#### Syntax

```
public System.String<CharT>& Append(const System.String<CharT>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.String<CharT>& | A string to append. |

#### Returns

System.String<CharT>&

Returns a reference to this string.

### Begin() Member Function

Returns an iterator to the beginning of the string.

### Syntax

```
public System.RandomAccessIter<CharT, CharT&, CharT*> Begin();
```

### Returns

System.RandomAccessIter<CharT, CharT&, CharT*>

Returns an iterator to the beginning of the string.

### Begin() const Member Function

Returns a constant iterator to the beginning of the string.

### Syntax

```
public System.RandomAccessIter<CharT, const CharT&, const CharT*> Begin() const;
```

### Returns

System.RandomAccessIter<CharT, const CharT&, const CharT*>

Returns a constant iterator to the beginning of the string.

### CBegin() const Member Function

Returns a constant iterator to the beginning of the string.

#### Syntax

```
public System.RandomAccessIter<CharT, const CharT&, const CharT*> CBegin() const;
```

#### Returns

System.RandomAccessIter<CharT, const CharT&, const CharT*>

Returns a constant iterator to the beginning of the string.

### CEnd() const Member Function

Returns a constant iterator one past the end of the string.

#### Syntax

```
public System.RandomAccessIter<CharT, const CharT&, const CharT*> CEnd() const;
```

#### Returns

System.RandomAccessIter<CharT, const CharT&, const CharT*>

Returns a constant iterator one past the end of the string.

### Capacity() const Member Function

Returns the number of characters that the string can hold without allocating more memory for it.

### Syntax

```
public int Capacity() const;
```

### Returns

int

Returns the number of characters that the string can hold without allocating more memory for it.

### Chars() const Member Function

Returns the string as a NUL-terminated sequence of characters.

#### Syntax

```
public const CharT* Chars() const;
```

#### Returns

const CharT*

Returns the string as a NUL-terminated sequence of characters.

### Clear() Member Function

Makes the string empty.

### Syntax

```
public void Clear();
```

### End() Member Function

Returns an iterator to one past the end of the string.

### Syntax

```
public System.RandomAccessIter<CharT, CharT&, CharT*> End();
```

### Returns

System.RandomAccessIter<CharT, CharT&, CharT*>

Returns an iterator to one past the end of the string.

### End() const Member Function

Returns a constant iterator to one past the end of the string.

#### Syntax

```
public System.RandomAccessIter<CharT, const CharT&, const CharT*> End() const;
```

#### Returns

System.RandomAccessIter<CharT, const CharT&, const CharT*>

Returns a constant iterator to one past the end of the string.

### EndsWith(const System.String<CharT>&) const Member Function

Returns true, if the string ends with the specified substring, false otherwise.

#### Syntax

```
public bool EndsWith(const System.String<CharT>& suffix) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| suffix | const System.String<CharT>& | A suffix to test. |

#### Returns

bool

Returns true, if the string ends with the specified substring, false otherwise.

#### Remarks

The comparison is case sensitive.

#### Example

```
using System;

//  Writes:
//  true

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.EndsWith("bush.") << endl();
}
```

### Find(CharT) const Member Function

Returns the index of the first occurrence of the given character within this string, or -1 if the specified character does not occur in this string.

#### Syntax

```
public int Find(CharT x) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | CharT | A character to search. |

#### Returns

int

Returns the index of the first occurrence of the given character within this string, or -1 if the specified character does not occur in this string.

#### Example

```
using System;

//  Writes:
//  10
//  -1

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.Find('t') << endl();
    Console.Out() << proverb.Find('x') << endl();
}
```

### Find(CharT, int) const Member Function

Returns the index of the first occurrence of the given character within this string, or -1 if the specified character does not occur in this string. The search starts with the specified index.

#### Syntax

```
public int Find(CharT x, int start) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | CharT | A character to search. |
| start | int | A search start index. |

#### Returns

int

Returns the index of the first occurrence of the given character within this string, or -1 if the specified character does not occur in this string. The search starts with the specified index.

#### Example

```
using System;

//  Writes:
//  25
//  -1

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.Find('t', 11) << endl();
    Console.Out() << proverb.Find('x', 11) << endl();
}
```

### Find(const System.String<CharT>&) const Member Function

Returns the starting index of the first occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string.

#### Syntax

```
public int Find(const System.String<CharT>& s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<CharT>& | A substring to search. |

#### Returns

int

Returns the starting index of the first occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string.

#### Example

```
using System;

//  Writes:
//  7
//  -1

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.Find("in") << endl();
    Console.Out() << proverb.Find("foo") << endl();
}
```

### Find(const System.String<CharT>&, int) const Member Function

Returns the starting index of the first occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string. The search starts with the specified index.

#### Syntax

```
public int Find(const System.String<CharT>& s, int start) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<CharT>& | A substring to search. |
| start | int | A search start index. |

#### Returns

int

Returns the starting index of the first occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string. The search starts with the specified index.

#### Example

```
using System;

//  Writes:
//  32
//  -1

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.Find("in", 8) << endl();
    Console.Out() << proverb.Find("foo", 8) << endl();
}
```

### IsEmpty() const Member Function

Returns true if the string is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the string is empty, false otherwise.

### Length() const Member Function

Returns the length of the string.

#### Syntax

```
public int Length() const;
```

#### Returns

int

Returns the length of the string.

### RFind(CharT) const Member Function

Returns the index of the last occurrence of the given character within this string, or -1 if the specified character does not occur in this string.

#### Syntax

```
public int RFind(CharT x) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | CharT | A character to search. |

#### Returns

int

Returns the index of the last occurrence of the given character within this string, or -1 if the specified character does not occur in this string.

#### Example

```
using System;

//  Writes:
//  35
//  -1

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.RFind('t') << endl();
    Console.Out() << proverb.RFind('x') << endl();
}
```

### RFind(CharT, int) const Member Function

Returns the index of the last occurrence of the given character within this string, or -1 if the specified character does not occur in this string. The search starts with the specified index.

#### Syntax

```
public int RFind(CharT x, int start) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | CharT | A character to search. |
| start | int | A search start index. |

#### Returns

int

Returns the index of the last occurrence of the given character within this string, or -1 if the specified character does not occur in this string. The search starts with the specified index.

#### Example

```
using System;

//  Writes:
//  28
//  -1

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.RFind('t', 34) << endl();
    Console.Out() << proverb.RFind('x', 34) << endl();
}
```

### RFind(const System.String<CharT>&) const Member Function

Returns the starting index of the last occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string.

#### Syntax

```
public int RFind(const System.String<CharT>& s) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<CharT>& | A substring to search. |

#### Returns

int

Returns the starting index of the last occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string.

#### Example

```
using System;

//  Writes:
//  32
//  -1

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.RFind("in") << endl();
    Console.Out() << proverb.RFind("foo") << endl();
}
```

### RFind(const System.String<CharT>&, int) const Member Function

Returns the starting index of the last occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string. The search starts with the specified index.

#### Syntax

```
public int RFind(const System.String<CharT>& s, int start) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<CharT>& | A substring to search. |
| start | int | A search start index. |

#### Returns

int

Returns the starting index of the last occurrence of the specified substring within this string, or -1 is the specified string does not occur within this string. The search starts with the specified index.

#### Example

```
using System;

//  Writes:
//  7
//  -1

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.RFind("in", 31) << endl();
    Console.Out() << proverb.RFind("foo", 31) << endl();
}
```

### Replace(CharT, CharT) Member Function

Replaces every occurrence of the given character with another character.

#### Syntax

```
public void Replace(CharT oldChar, CharT newChar);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| oldChar | CharT | A character to replace. |
| newChar | CharT | The replacement character. |

#### Example

```
using System;

//  Writes:
//  A bord on the hand os worth two on the bush.

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    proverb.Replace('i', 'o');
    Console.Out() << proverb << endl();
}
```

### Reserve(int) Member Function

Reserves room for a string with the given number of characters.

#### Syntax

```
public void Reserve(int minLen);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| minLen | int | The minimum number of characters that the string can hold without a memory allocation. |

### Split(CharT) Member Function

Returns a list of substrings separated by the given character.

#### Syntax

```
public System.Collections.List<System.String<CharT>> Split(CharT c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | CharT | A separator character. |

#### Returns

System.Collections.List<System.String<CharT>>

Returns a list of substrings separated by the given character.

#### Example

```
using System;
using System.Collections;

//  Writes:
//  A
//  bird
//  in
//  the
//  hand
//  is
//  worth
//  two
//  in
//  the
//  bush.

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    List<string> words = proverb.Split(' ');
    for (const string& word : words)
    {
        Console.Out() << word << endl();
    }
}
```

### StartsWith(const System.String<CharT>&) const Member Function

Returns true if the string starts with the given substring, false otherwise.

#### Syntax

```
public bool StartsWith(const System.String<CharT>& prefix) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| prefix | const System.String<CharT>& | A prefix to test. |

#### Returns

bool

Returns true if the string starts with the given substring, false otherwise.

### String<CharT>(CharT) Member Function

Constructor. Constructs a string that has the given character.

#### Syntax

```
public String<CharT>(CharT c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | CharT | A character. |

### String<CharT>(CharT, int) Member Function

Constructor. Constructs a string that has given number of the specified character.

#### Syntax

```
public String<CharT>(CharT c, int n);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | CharT | A character. |
| n | int | Number of characters. |

#### Example

```
using System;

//  Writes:
//  aaaaaa

void main()
{
    string s('a', 6);
    Console.Out() << s << endl();
}
```

### String<CharT>(const CharT*) Member Function

Constructor. Constructs a string from a NUL-terminated sequence of characters.

#### Syntax

```
public String<CharT>(const CharT* chars_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| chars_ | const CharT* | A NUL-terminated sequence of characters. |

### String<CharT>(const CharT*, const CharT*) Member Function

Constructor. Constructs a string of characters included in range begin and end.

#### Syntax

```
public String<CharT>(const CharT* begin, const CharT* end);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | const CharT* | Start of the character range. |
| end | const CharT* | One past the end of the character range. |

### String<CharT>(const CharT*, int) Member Function

Constructor. Constructs a string from given number of characters of a NUL-terminated sequence of characters.

#### Syntax

```
public String<CharT>(const CharT* chars_, int length_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| chars_ | const CharT* | A NUL-terminated sequence of characters. |
| length_ | int | Maximum number of characters to copy. |

#### Example

```
using System;

//  Writes:
//  A bird

void main()
{
    string proverb("A bird in the hand is worth two in the bush.", 6);
    Console.Out() << proverb << endl();
}
```

### Substring(int) const Member Function

Returns a substring starting from the given index.

#### Syntax

```
public System.String<CharT> Substring(int start) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| start | int | A starting index of the substring. |

#### Returns

System.String<CharT>

Returns a substring starting from the given index.

#### Example

```
using System;

//  Writes:
//  worth two in the bush.

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.Substring(22) << endl();
}
```

### Substring(int, int) const Member Function

Returns a substring starting from the given index whose length is at most given number of characters.

#### Syntax

```
public System.String<CharT> Substring(int start, int length) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| start | int | A starting index of the substring. |
| length | int | The maximum number of characters in the substring. |

#### Returns

System.String<CharT>

Returns a substring starting from the given index whose length is at most given number of characters.

#### Example

```
using System;

//  Writes:
//  worth
//  bush.

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    Console.Out() << proverb.Substring(22,5) << endl();
    Console.Out() << proverb.Substring(39,10) << endl();
}
```

### operator<(const System.String<CharT>&) const Member Function

Compares a string for less than relationship with another string.

#### Syntax

```
public bool operator<(const System.String<CharT>& that) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.String<CharT>& | A string to compare with. |

#### Returns

bool

Returns true if this string comes lexicographically before the specified string, false otherwise.

#### Remarks

The comparison is case sensitive and done with the character code values of the characters.

### operator==(const System.String<CharT>&) const Member Function

Compares a string for equality with another string.

#### Syntax

```
public bool operator==(const System.String<CharT>& that) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.String<CharT>& | A string to compare with. |

#### Returns

bool

Returns true if this string has the same number of characters than the given string, and the characters are pairwise equal, false otherwise.

#### Remarks

The comparison is case sensitive.

### operator[](int) Member Function

Returns a reference to the character with the specified index.

#### Syntax

```
public CharT& operator[](int index);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| index | int | An index. |

#### Returns

CharT&

Returns a reference to the character with the specified index.

#### Example

```
using System;

//  Writes:
//  A bird in the band is worth two in the bush.

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    proverb[14] = 'b';
    Console.Out() << proverb << endl();
}
```

### operator[](int) const Member Function

Returns a character with the specified index.

#### Syntax

```
public CharT operator[](int index) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| index | int | An index. |

#### Returns

CharT

Returns a character with the specified index.

#### ~String<CharT>() Member Function

Destructor. Releases the memory occupied by the string.

**Syntax**

```
public ~String<CharT>();
```

### 2.2.56.3 Nonmember Functions

| Function | Description |
|---|---|
| operator+(const CharT*, const System.String<-CharT>&) | Concatenates a NUL-terminated string and a string. |
| operator+(const System.String<CharT>&, const CharT*) | Concatenates a string and a NUL-terminated string. |
| operator+(const System.String<CharT>&, const System.String<CharT>&) | Concatenates two strings. |

### operator+(const CharT*, const System.String<CharT>&) Function

Concatenates a NUL-terminated string and a string.

#### Syntax

```
public System.String<CharT> operator+(const CharT* first, const System.String<CharT>&
second);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| first | const CharT* | A NUL-terminated string. |
| second | const System.String<CharT>& | A string. |

#### Returns

System.String<CharT>

Returns a string that result when *first* and *second* is concatenated.

### operator+(const System.String<CharT>&, const CharT*) Function

Concatenates a string and a NUL-terminated string.

#### Syntax

```
public System.String<CharT> operator+(const System.String<CharT>& first, const CharT*
second);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| first | const System.String<CharT>& | A string. |
| second | const CharT* | A NUL-terminated sequence of characters. |

#### Returns

System.String<CharT>

Returns a string that result when *first* and *second* is concatenated.

#### Example

```
using System;

//  Writes:
//  A bird in the hand is worth two in the bush.

void main()
{
    const char* start = "A bird in the hand is worth ";
    string end("two in the bush.");
    string proverb = start + end;
    Console.Out() << proverb << endl();
}
```

### operator+(const System.String<CharT>&, const System.String<CharT>&) Function

Concatenates two strings.

#### Syntax

```
public System.String<CharT> operator+(const System.String<CharT>& first, const System.String<Cl
second);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| first | const System.String<CharT>& | The first string. |
| second | const System.String<CharT>& | The second string. |

#### Returns

System.String<CharT>

Returns a string that result when *first* and *second* is concatenated.

#### Example

```
using System;

//  Writes:
//  A bird in the hand is worth two in the bush.

void main()
{
    string start("A bird in the hand is worth ");
    string end("two in the bush.");
    string proverb = start + end;
    Console.Out() << proverb << endl();
}
```

## 2.2.57 TimeError Class

An exception thrown when a time function fails.

### Syntax

```
public class TimeError;
```

### Base Class

System.Exception

#### 2.2.57.1 Member Functions

| Member Function | Description |
|---|---|
| TimeError() | Default constructor. |
| TimeError(const System.TimeError&) | Copy constructor. |
| operator=(const System.TimeError&) | Copy assignment. |
| TimeError(System.TimeError&&) | Move constructor. |
| operator=(System.TimeError&&) | Move assignment. |
| TimeError(const System.String<char>&, const System.String<char>&) | Constructor. Initializes the time error with specified operation description and failure reason. |
| ~TimeError() | Destructor. |

### TimeError() Member Function

Default constructor.

#### Syntax

```
public TimeError();
```

### TimeError(const System.TimeError&) Member Function

Copy constructor.

#### Syntax

```
public TimeError(const System.TimeError& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.TimeError& | Argument to copy. |

### operator=(const System.TimeError&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.TimeError& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.TimeError& | Argument to assign. |

### TimeError(System.TimeError&&) Member Function

Move constructor.

#### Syntax

```
public TimeError(System.TimeError&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.TimeError&& | Argument to move from. |

### operator=(System.TimeError&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.TimeError&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.TimeError&& | Argument to assign from. |

### TimeError(const System.String<char>&, const System.String<char>&) Member Function

Constructor. Initializes the time error with specified operation description and failure reason.

#### Syntax

```
public TimeError(const System.String<char>& operation, const System.String<char>&
reason);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| operation | const System.String<char>& | Description of operation. |
| reason | const System.String<char>& | Failure reason. |

### ~TimeError() Member Function

Destructor.

#### Syntax

```
public ~TimeError();
```

## 2.2.58 TimePoint Class

Represents a point in time as specified nanoseconds elapsed since epoch.

### Syntax

```
public class TimePoint;
```

### 2.2.58.1 Remarks

Epoch is midnight 1.1.1970.

### 2.2.58.2 Member Functions

| Member Function | Description |
|---|---|
| TimePoint() | Constructor. Initializes the time point to zero nanoseconds elapsed since epoch. |
| TimePoint(const System.TimePoint&) | Copy constructor. |
| operator=(const System.TimePoint&) | Copy assignment. |
| TimePoint(System.TimePoint&&) | Move constructor. |
| operator=(System.TimePoint&&) | Move assignment. |
| Rep() const | Returns the number of nanoseconds elapsed since epoch. |
| TimePoint(long) | Constructor. Initializes the time point with the given number of nanoseconds elapsed since epoch. |

### TimePoint() Member Function

Constructor. Initializes the time point to zero nanoseconds elapsed since epoch.

#### Syntax

```
public TimePoint();
```

### TimePoint(const System.TimePoint&) Member Function

Copy constructor.

#### Syntax

```
public TimePoint(const System.TimePoint& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.TimePoint& | Argument to copy. |

### operator=(const System.TimePoint&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.TimePoint& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.TimePoint& | Argument to assign. |

### TimePoint(System.TimePoint&&) Member Function

Move constructor.

#### Syntax

```
public TimePoint(System.TimePoint&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.TimePoint&& | Argument to move from. |

### operator=(System.TimePoint&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.TimePoint&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.TimePoint&& | Argument to assign from. |

### Rep() const Member Function

Returns the number of nanoseconds elapsed since epoch.

### Syntax

```
public long Rep() const;
```

### Returns

long

Returns the number of nanoseconds elapsed since epoch.

#### TimePoint(long) Member Function

Constructor. Initializes the time point with the given number of nanoseconds elapsed since epoch.

#### Syntax

```
public TimePoint(long nanosecs_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| nanosecs_ | long | Number of nanoseconds. |

### 2.2.58.3 Nonmember Functions

| Function | Description |
|---|---|
| operator+(System.TimePoint, System.Duration) | Computes the sum of a time point and a duration and returns a time point. |
| operator-(System.TimePoint, System.Duration) | Subtracts a duration from a time point and returns a time point. |
| operator-(System.TimePoint, System.TimePoint) | Subtracts a time point from a time point and returns a duration. |
| operator<(System.TimePoint, System.TimePoint) | Compares two time points for less than relationship. |
| operator==(System.TimePoint, System.TimePoint) | Compares two time points for equality. |

### operator+(System.TimePoint, System.Duration) Function

Computes the sum of a time point and a duration and returns a time point.

#### Syntax

```
public System.TimePoint operator+(System.TimePoint tp, System.Duration d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| tp | System.TimePoint | A time point. |
| d | System.Duration | A duration. |

#### Returns

System.TimePoint

Returns tp+d.

### operator-(System.TimePoint, System.Duration) Function

Subtracts a duration from a time point and returns a time point.

#### Syntax

```
public System.TimePoint operator-(System.TimePoint tp, System.Duration d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| tp | System.TimePoint | A time point. |
| d | System.Duration | A duration. |

#### Returns

System.TimePoint

Returns $TimePoint(tp - d)$.

### operator-(System.TimePoint, System.TimePoint) Function

Subtracts a time point from a time point and returns a duration.

#### Syntax

```
public System.Duration operator-(System.TimePoint left, System.TimePoint right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.TimePoint | The first time point. |
| right | System.TimePoint | The second time point. |

#### Returns

System.Duration

Returns $Duration(left - right)$.

### operator<(System.TimePoint, System.TimePoint) Function

Compares two time points for less than relationship.

#### Syntax

```
public bool operator<(System.TimePoint left, System.TimePoint right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.TimePoint | The first time point. |
| right | System.TimePoint | The second time point. |

#### Returns

bool

Returns true, if the first time point is less than the second time point, false otherwise.

### operator==(System.TimePoint, System.TimePoint) Function

Compares two time points for equality.

#### Syntax

```
public bool operator==(System.TimePoint left, System.TimePoint right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.TimePoint | The first time point. |
| right | System.TimePoint | The second time point. |

#### Returns

bool

Returns true, if the first time point is equal to the second time point, false otherwise.

## 2.2.59 TracedFun Class

Function tracer.

### Syntax

```
public class TracedFun;
```

#### 2.2.59.1 Member Functions

| Member Function | Description |
|---|---|
| TracedFun() | Default constructor. |
| TracedFun(const char*, const char*, int) | Writes > followed by the given function name, file and line to stardard error stream. |
| ~TracedFun() | Writes < followed by the function name, file and line to stardard error stream. |

### TracedFun() Member Function

Default constructor.

#### Syntax

```
public TracedFun();
```

### TracedFun(const char*, const char*, int) Member Function

Writes > followed by the given function name, file and line to stardard error stream.

#### Syntax

```
public TracedFun(const char* fun_, const char* file_, int line_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fun_ | const char* | A name of a function. |
| file_ | const char* | Source file name. |
| line_ | int | Source line number. |

### ~TracedFun() Member Function

Writes < followed by the function name, file and line to stardard error stream.

#### Syntax

```
public ~TracedFun();
```

## 2.2.60 Tracer Class

A utility class for tracing entry and exit of some operation.

### Syntax

```
public class Tracer;
```

### 2.2.60.1 Member Functions

| Member Function | Description |
|---|---|
| Tracer() | Default constructor. |
| Tracer(const System.String<char>&) | Constructor. Writes the specified string to standard error stream. |
| ~Tracer() | Destructor. Writes ~ and a string specified in constructor to standard error stream. |

### Tracer() Member Function

Default constructor.

### Syntax

```
public Tracer();
```

### Tracer(const System.String<char>&) Member Function

Constructor. Writes the specified string to standard error stream.

#### Syntax

```
public Tracer(const System.String<char>& s_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s_ | const System.String<char>& | String to write. |

### ~Tracer() Member Function

Destructor. Writes ~ and a string specified in constructor to standard error stream.

#### Syntax

```
public ~Tracer();
```

## 2.2.61 UnaryFun<Argument, Result> Class

A base class for unary function objects.

### Syntax

```
public class UnaryFun<Argument, Result>;
```

### Constraint

Argument is Semiregular

#### 2.2.61.1 Remarks

A derived unary function inherits the type definitions of this base class and provides an implementation for the *operator()(ArgumentType)* function.

#### 2.2.61.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| ArgumentType | Argument | The type of the argument of the unary function. |
| ResultType | Result | The type of the result of the unary function. |

#### 2.2.61.3 Member Functions

| Member Function | Description |
|---|---|
| UnaryFun<Argument, Result>() | Default constructor. |
| UnaryFun<Argument, Result>(const System.-UnaryFun<Argument, Result>&) | Copy constructor. |
| operator=(const System.UnaryFun<Argument, Result>&) | Copy assignment. |
| UnaryFun<Argument, Result>(System.-UnaryFun<Argument, Result>&&) | Move constructor. |
| operator=(System.UnaryFun<Argument, Result>&&) | Move assignment. |

### UnaryFun<Argument, Result>() Member Function

Default constructor.

#### Syntax

```
public UnaryFun<Argument, Result>();
```

### UnaryFun<Argument, Result>(const System.UnaryFun<Argument, Result>&) Member Function

Copy constructor.

#### Syntax

```
public UnaryFun<Argument, Result>(const System.UnaryFun<Argument, Result>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.UnaryFun<Argument, Result>& | Argument to copy. |

### operator=(const System.UnaryFun<Argument, Result>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.UnaryFun<Argument, Result>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.UnaryFun<Argument, Result>& | Argument to assign. |

### UnaryFun<Argument, Result>(System.UnaryFun<Argument, Result>&&) Member Function

Move constructor.

#### Syntax

```
public UnaryFun<Argument, Result>(System.UnaryFun<Argument, Result>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.UnaryFun<Argument, Result>&& | Argument to move from. |

### operator=(System.UnaryFun<Argument, Result>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.UnaryFun<Argument, Result>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.UnaryFun<Argument, Result>&& | Argument to assign from. |

## 2.2.62 UnaryPred<Argument> Class

A base class for unary predicates.

### Syntax

```
public class UnaryPred<Argument>;
```

### Constraint

Argument is Semiregular

### Model of

UnaryPredicate<T>

### Base Class

System.UnaryFun<Argument, bool>

#### 2.2.62.1 Remarks

A unary predicate is an unary function whose application operator returns a truth value.

#### 2.2.62.2 Member Functions

| Member Function | Description |
|---|---|
| UnaryPred<Argument>() | Default constructor. |
| UnaryPred<Argument>(const System.UnaryPred<Argument>&) | Copy constructor. |
| operator=(const System.UnaryPred<Argument>&) | Copy assignment. |
| UnaryPred<Argument>(System.UnaryPred<Argument>&&) | Move constructor. |
| operator=(System.UnaryPred<Argument>&&) | Move assignment. |

### UnaryPred<Argument>() Member Function

Default constructor.

#### Syntax

```
public UnaryPred<Argument>();
```

### UnaryPred<Argument>(const System.UnaryPred<Argument>&) Member Function

Copy constructor.

#### Syntax

```
public UnaryPred<Argument>(const System.UnaryPred<Argument>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.UnaryPred<Argument>& | Argument to copy. |

### operator=(const System.UnaryPred<Argument>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.UnaryPred<Argument>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.UnaryPred<Argument>& | Argument to assign. |

### UnaryPred<Argument>(System.UnaryPred<Argument>&&) Member Function

Move constructor.

#### Syntax

```
public UnaryPred<Argument>(System.UnaryPred<Argument>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.UnaryPred<Argument>&& | Argument to move from. |

### operator=(System.UnaryPred<Argument>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.UnaryPred<Argument>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.UnaryPred<Argument>&& | Argument to assign from. |

## 2.2.63 UniquePtr<T> Class

A unique pointer to an object.

### Syntax

```
public class UniquePtr<T>;
```

### Model of

DefaultConstructible<T>

Movable<T>

#### 2.2.63.1 Remarks

The unique pointer destroys the object it owns in its destructor. The copy constructor and copy assignment operator are suppressed, but unique pointer has move constructor and move assignment operator, so it can be moved to containers.

#### 2.2.63.2 Example

```
using System;
using System.Collections;

//  Writes:
//  foo
//  bar

void main()
{
    List<UniquePtr<string>> list;
    UniquePtr<string> foo(new string("foo"));
    list.Add(Rvalue(foo));
    list.Add(UniquePtr<string>(new string("bar")));
    for (const UniquePtr<string>& s : list)
    {
        Console.Out() << *s << endl();
    }
}
```

#### 2.2.63.3 Member Functions

| Member Function | Description |
|---|---|
| UniquePtr<T>() | Constructor. Constructs a null unique pointer. |
| UniquePtr<T>(System.UniquePtr<T>&&) | Move constructor. |
| operator=(System.UniquePtr<T>&&) | Move assignment. |
| GetPtr() const | Returns the contained pointer to the owned object. |

| | |
|---|---|
| IsNull() const | Returns true if the contained pointer is null, false otherwise. |
| Release() | Releases the ownership of the owned object and sets the unique pointer to null. |
| Reset() | Resets the unique ptr to null. |
| Reset(T*) | Resets the contained pointer to point to a new object. |
| Swap(System.UniquePtr<T>&) | Exchanges the contents of the unique pointer with another unique pointer. |
| UniquePtr<T>(T*) | Constructor. Constructs a unique pointer to the given object. |
| operator*() const | Returns a reference to the pointed object. |
| operator->() const | Returns the contained pointer to the pointed object. |
| operator=(T*) | Assigns a new object to the unique pointer. |
| ~UniquePtr<T>() | Destructor. Destroys the owned object. |

### UniquePtr<T>() Member Function

Constructor. Constructs a null unique pointer.

#### Syntax

```
public UniquePtr<T>();
```

### UniquePtr<T>(System.UniquePtr<T>&&) Member Function

Move constructor.

#### Syntax

```
public UniquePtr<T>(System.UniquePtr<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.UniquePtr<T>&& | A unique pointer to move from. |

### operator=(System.UniquePtr<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.UniquePtr<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.UniquePtr<T>&& | A unique pointer to move from. |

### GetPtr() const Member Function

Returns the contained pointer to the owned object.

#### Syntax

```
public T* GetPtr() const;
```

#### Returns

T*

Returns the contained pointer to the owned object.

### IsNull() const Member Function

Returns true if the contained pointer is null, false otherwise.

#### Syntax

```
public bool IsNull() const;
```

#### Returns

bool

Returns true if the contained pointer is null, false otherwise.

### Release() Member Function

Releases the ownership of the owned object and sets the unique pointer to null.

### Syntax

```
public T* Release();
```

### Returns

T*

Returns the contained pointer.

### Reset() Member Function

Resets the unique ptr to null.

### Syntax

```
public void Reset();
```

### Remarks

Destroys the owned object.

### Reset(T*) Member Function

Resets the contained pointer to point to a new object.

#### Syntax

```
public void Reset(T* ptr_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to an object. |

#### Remarks

If the unique pointer owns an object before the operation, destroys the owned object before the operation. Then acquires the ownership of the new object.

### Swap(System.UniquePtr<T>&) Member Function

Exchanges the contents of the unique pointer with another unique pointer.

#### Syntax

```
public void Swap(System.UniquePtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.UniquePtr<T>& | A unique pointer to exchange contents with. |

### UniquePtr<T>(T*) Member Function

Constructor. Constructs a unique pointer to the given object.

#### Syntax

```
public UniquePtr<T>(T* ptr_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to an object. |

### operator*() const Member Function

Returns a reference to the pointed object.

#### Syntax

```
public T& operator*() const;
```

#### Returns

T&

Returns a reference to the pointed object.

### operator->() const Member Function

Returns the contained pointer to the pointed object.

#### Syntax

```
public T* operator->() const;
```

#### Returns

T*

Returns the contained pointer to the pointed object.

### operator=(T*) Member Function

Assigns a new object to the unique pointer.

#### Syntax

```
public void operator=(T* ptr_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to an object. |

#### Remarks

If the unique pointer is not null before the assignment, destroys the old object before the assignment. Then acquires the ownership of the new object.

#### ~UniquePtr<T>() Member Function

Destructor. Destroys the owned object.

**Syntax**

```
public ~UniquePtr<T>();
```

### 2.2.63.4 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.UniquePtr<T>&, const System.UniquePtr<T>&) | Compares two unique pointers for less than relationship. |
| operator==(const System.UniquePtr<T>&, const System.UniquePtr<T>&) | Compares two unique pointers for equality. |

### operator<(const System.UniquePtr<T>&, const System.UniquePtr<T>&) Function

Compares two unique pointers for less than relationship.

#### Syntax

```
public bool operator<(const System.UniquePtr<T>& left, const System.UniquePtr<T>&
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.UniquePtr<T>& | The first unique pointer. |
| right | const System.UniquePtr<T>& | The second unique pointer. |

#### Returns

bool

Returns true if the pointer contained by the first unique pointer is less than the pointer contained by the second unique pointer, false otherwise.

### operator==(const System.UniquePtr<T>&, const System.UniquePtr<T>&) Function

Compares two unique pointers for equality.

#### Syntax

```
public bool operator==(const System.UniquePtr<T>& left, const System.UniquePtr<T>&
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.UniquePtr<T>& | The first unique pointer. |
| right | const System.UniquePtr<T>& | The second unique pointer. |

#### Returns

bool

Returns true if the pointer contained by the first unique pointer is equal to the pointer contained by the second unique pointer.

## 2.2.64 WeakCount<T> Class

A handle to a Counter<T> that maintains the weak count portion of the counter.

### Syntax

```
public class WeakCount<T>;
```

### 2.2.64.1 Member Functions

| Member Function | Description |
|---|---|
| WeakCount<T>() | Constructor. Initializes an empty weak count. |
| WeakCount<T>(const System.WeakCount<T>-&) | Copy constructor. Increments weak count. |
| operator=(const System.WeakCount<T>&) | Copy assignment. Decrements the weak count of the old counter and increments the weak count of the copied counter. |
| GetCounter() const | Returns the contained pointer to a counter. |
| GetUseCount() const | Returns the use count. |
| Swap(System.WeakCount<T>&) | Exchanges the contents of the weak count with another weak count. |
| WeakCount<T>(const System.SharedCount<-T>&) | Constructor. Constructs a weak count from a shared count. |
| operator=(const System.SharedCount<T>&) | Assignment. Decrements the weak count of the old counter and increments the weak count of the copied counter. |
| ~WeakCount<T>() | Destructor. Decrements the weak count. |

### WeakCount<T>() Member Function

Constructor. Initializes an empty weak count.

#### Syntax

```
public WeakCount<T>();
```

### WeakCount<T>(const System.WeakCount<T>&) Member Function

Copy constructor. Increments weak count.

#### Syntax

```
public WeakCount<T>(const System.WeakCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.WeakCount<T>& | A weak count to copy. |

### operator=(const System.WeakCount<T>&) Member Function

Copy assignment. Decrements the weak count of the old counter and increments the weak count of the copied counter.

#### Syntax

```
public void operator=(const System.WeakCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.WeakCount<T>& | A weak count to assign. |

### GetCounter() const Member Function

Returns the contained pointer to a counter.

#### Syntax

```
public System.Counter<T>* GetCounter() const;
```

#### Returns

System.Counter<T>*

Returns the contained pointer to a counter.

### GetUseCount() const Member Function

Returns the use count.

#### Syntax

```
public int GetUseCount() const;
```

#### Returns

int

Returns the use count.

### Swap(System.WeakCount<T>&) Member Function

Exchanges the contents of the weak count with another weak count.

#### Syntax

```
public void Swap(System.WeakCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.WeakCount<T>& | A weak count to exchange contents with. |

### WeakCount<T>(const System.SharedCount<T>&) Member Function

Constructor. Constructs a weak count from a shared count.

#### Syntax

```
public WeakCount<T>(const System.SharedCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SharedCount<T>& | A shared count. |

### operator=(const System.SharedCount<T>&) Member Function

Assignment. Decrements the weak count of the old counter and increments the weak count of the copied counter.

#### Syntax

```
public void operator=(const System.SharedCount<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SharedCount<T>& | A shared count to assign. |

#### ~WeakCount<T>() Member Function

Destructor. Decrements the weak count.

**Syntax**

```
public ~WeakCount<T>();
```

### 2.2.64.2 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.WeakCount<T>&, const System.WeakCount<T>&) | Compares two weak counts for less than relationship. |
| operator==(const System.WeakCount<T>&, const System.WeakCount<T>&) | Compares two weak counts for equality. |

### operator<(const System.WeakCount<T>&, const System.WeakCount<T>&) Function

Compares two weak counts for less than relationship.

#### Syntax

```
public bool operator<(const System.WeakCount<T>& left, const System.WeakCount<T>&
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.WeakCount<T>& | The first weak count. |
| right | const System.WeakCount<T>& | The second weak count. |

#### Returns

bool

Returns true if the memory address of the counter contained by the *left* count is less than the memory addess of the counter contained by the *right* count, false otherwise.

### operator==(const System.WeakCount<T>&, const System.WeakCount<T>&) Function

Compares two weak counts for equality.

#### Syntax

```
public bool operator==(const System.WeakCount<T>& left, const System.WeakCount<T>&
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.WeakCount<T>& | The first weak count. |
| right | const System.WeakCount<T>& | The second weak count. |

#### Returns

bool

Returns true if *left* contains the same counter as *right* or both are empty, false otherwise.

## 2.2.65 WeakPtr<T> Class

Used to break cycles in shared ownership.

### Syntax

```
public class WeakPtr<T>;
```

### 2.2.65.1 Remarks

If objects contain shared pointers to each other thus forming a cycle, the use count will never go to zero, and the destructors of the objects will not be called. Replacing one of the shared pointers with a weak pointer breaks the cycle and the objects will be released.

### 2.2.65.2 Example

```
using System;
using System.Collections;

//  Writes:
//  a.next == b
//  b.next == c
//  c.next == a
//  a destroyed
//  b destroyed
//  c destroyed

public typedef SharedPtr<Base> BasePtr;
public typedef WeakPtr<Base> WeakBasePtr;

public abstract class Base
{
    public Base(const string& name_): name(name_)
    {
    }
    public virtual ~Base()
    {
        Console.WriteLine(name + " destroyed");
    }
    public const string& Name() const
    {
        return name;
    }
    public abstract BasePtr GetNext() const;
    public abstract void SetNext(BasePtr next_);
    public void PrintNext()
    {
        BasePtr next = GetNext();
        if (!next.IsNull())
        {
            Console.WriteLine(name + ".next == " + next->Name());
        }
    }
    private string name;
}

public class A: Base
```

```
{
    public A(const string& name_): base(name_)
    {
    }
    public override BasePtr GetNext() const
    {
        return next;
    }
    public override void SetNext(BasePtr next_)
    {
        next = next_;
    }
    private BasePtr next;
}

public class B: Base
{
    public B(const string& name_): base(name_)
    {
    }
    public override BasePtr GetNext() const
    {
        return next;
    }
    public override void SetNext(BasePtr next_)
    {
        next = next_;
    }
    private BasePtr next;
}

public class C: Base
{
    public C(const string& name_): base(name_)
    {
    }
    public override BasePtr GetNext() const
    {
        return next.Lock();
    }
    public override void SetNext(BasePtr next_)
    {
        next = next_;
    }
    private WeakBasePtr next;
}


void main()
{
    List<BasePtr> objects;
    BasePtr a(new A("a"));
    BasePtr b(new B("b"));
    BasePtr c(new C("c"));
    a->SetNext(b);
    b->SetNext(c);
    c->SetNext(a);
    objects.Add(a);
```

```
    objects.Add(b);
    objects.Add(c);
    for (BasePtr o : objects)
    {
        o->PrintNext();
    }
}
```

### 2.2.65.3 Member Functions

| Member Function | Description |
|---|---|
| WeakPtr<T>() | Constructor. Constructs null weak pointer. |
| WeakPtr<T>(const System.WeakPtr<T>&) | Copy constructor. Increments weak count. |
| WeakPtr<T>(const System.WeakPtr<T>&) | Copy constructor. Increments weak count. |
| operator=(const System.WeakPtr<T>&) | Copy assignment. |
| operator=(const System.WeakPtr<T>&) | Copy assignment. |
| Assign(T*, const System.SharedCount<T>&) | Implementation detail to support System.-EnableSharedFromThis.T.U.System.-ShareableFromThis.T.ptr.U.ptr.System.-SharedCount.U.const.ref function. |
| GetCount() const | Returns the weak count. |
| GetPtr() const | Returns a pointer to the counted object. |
| GetUseCount() const | Returns the use count of the counted object. |
| IsExpired() const | Returns true if the use count has gone to zero and the counted object has been destroyed. |
| Lock() const | If the weak pointer has not been expired, returns a shared pointer to the counted object; otherwise returns null shared pointer. |
| Reset() | Resets the weak pointer to null. |
| Swap(System.WeakPtr<T>&) | Exchanges the contents of this weak pointer with another. |
| WeakPtr<T>(const System.SharedPtr<T>&) | Constructor. Constructs a weak pointer from a shared pointer. |
| operator=(const System.SharedPtr<T>&) | Assignment. Assigns a weak pointer from a shared pointer. |

| | |
|---|---|
| ~WeakPtr<T>() | Destructor. Decrements weak count. |

### WeakPtr<T>() Member Function

Constructor. Constructs null weak pointer.

#### Syntax

```
public WeakPtr<T>();
```

### WeakPtr<T>(const System.WeakPtr<T>&) Member Function

Copy constructor. Increments weak count.

#### Syntax

```
public WeakPtr<T>(const System.WeakPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.WeakPtr<T>& | A weak pointer to copy. |

### WeakPtr<T>(const System.WeakPtr<T>&) Member Function

Copy constructor. Increments weak count.

#### Syntax

```
public WeakPtr<T>(const System.WeakPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.WeakPtr<T>& | A weak pointer to copy. |

### operator=(const System.WeakPtr<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.WeakPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.WeakPtr<T>& | A weak pointer to assign. |

### operator=(const System.WeakPtr<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.WeakPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.WeakPtr<T>& | A weak pointer to assign. |

### Assign(T*, const System.SharedCount<T>&) Member Function

Implementation detail to support System.EnableSharedFromThis.T.U.System.-ShareableFromThis.T.ptr.U.ptr.System.SharedCount.U.const.ref function.

#### Syntax

```
public void Assign(T* ptr_, const System.SharedCount<T>& count_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| ptr_ | T* | A pointer to counted object. |
| count_ | const System.SharedCount<T>& | A shared count. |

### GetCount() const Member Function

Returns the weak count.

#### Syntax

```
public const System.WeakCount<T>& GetCount() const;
```

#### Returns

const System.WeakCount<T>&

Returns the weak count.

### GetPtr() const Member Function

Returns a pointer to the counted object.

#### Syntax

```
public T* GetPtr() const;
```

#### Returns

T*

Returns a pointer to the counted object.

### GetUseCount() const Member Function

Returns the use count of the counted object.

#### Syntax

```
public int GetUseCount() const;
```

#### Returns

int

Returns the use count of the counted object.

### IsExpired() const Member Function

Returns true if the use count has gone to zero and the counted object has been destroyed.

#### Syntax

```
public bool IsExpired() const;
```

#### Returns

bool

Returns true if the use count has gone to zero and the counted object has been destroyed.

### Lock() const Member Function

If the weak pointer has not been expired, returns a shared pointer to the counted object; otherwise returns null shared pointer.

### Syntax

```
public System.SharedPtr<T> Lock() const;
```

### Returns

System.SharedPtr<T>

If the weak pointer has not been expired, returns a shared pointer to the counted object; otherwise returns null shared pointer.

### Reset() Member Function

Resets the weak pointer to null.

### Syntax

```
public void Reset();
```

### Swap(System.WeakPtr<T>&) Member Function

Exchanges the contents of this weak pointer with another.

#### Syntax

```
public void Swap(System.WeakPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.WeakPtr<T>& | A weak pointer to exchange contents with. |

### WeakPtr<T>(const System.SharedPtr<T>&) Member Function

Constructor. Constructs a weak pointer from a shared pointer.

#### Syntax

```
public WeakPtr<T>(const System.SharedPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SharedPtr<T>& | A shared pointer. |

### operator=(const System.SharedPtr<T>&) Member Function

Assignment. Assigns a weak pointer from a shared pointer.

#### Syntax

```
public void operator=(const System.SharedPtr<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.SharedPtr<T>& | A shared pointer to assign. |

### ~WeakPtr<T>() Member Function

Destructor. Decrements weak count.

#### Syntax

```
public ~WeakPtr<T>();
```

## 2.2.66 uhuge Class

128-bit unsigned integer type.

### Syntax

```
public class uhuge;
```

### 2.2.66.1 Member Functions

| Member Function | Description |
|---|---|
| uhuge() | Constructor. Initializes the value to zero. |
| uhuge(const System.uhuge&) | Copy constructor. |
| operator=(const System.uhuge&) | Copy assignment. |
| uhuge(System.uhuge&&) | Move constructor. |
| operator=(System.uhuge&&) | Move assignment. |
| operator++() | Increments the value by one. |
| operator–() | Decrements the value by one. |
| uhuge(ulong) | Constructor. Initializes the value to specified 64-bit value. |
| uhuge(ulong, ulong) | Constructor. Initializes the value to $2^{64}h_ + l_$. |

### uhuge() Member Function

Constructor. Initializes the value to zero.

### Syntax

```
public uhuge();
```

### uhuge(const System.uhuge&) Member Function

Copy constructor.

#### Syntax

```
public uhuge(const System.uhuge& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.uhuge& | Argument to copy. |

### operator=(const System.uhuge&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.uhuge& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.uhuge& | Argument to assign. |

### uhuge(System.uhuge&&) Member Function

Move constructor.

#### Syntax

```
public uhuge(System.uhuge&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.uhuge&& | Argument to move from. |

### operator=(System.uhuge&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.uhuge&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.uhuge&& | Argument to assign from. |

### operator++() Member Function

Increments the value by one.

#### Syntax

```
public System.uhuge& operator++();
```

#### Returns

System.uhuge&

Returns the value.

### operator–() Member Function

Decrements the value by one.

#### Syntax

```
public System.uhuge& operator--();
```

#### Returns

System.uhuge&

Returns the value.

### uhuge(ulong) Member Function

Constructor. Initializes the value to specified 64-bit value.

#### Syntax

```
public uhuge(ulong l_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l_ | ulong | An unsigned 64-bit value. |

**uhuge(ulong, ulong) Member Function**

Constructor. Initializes the value to $2^{64}h_ + l_$.

**Syntax**

```
public uhuge(ulong h_, ulong l_);
```

**Parameters**

| Name | Type | Description |
|---|---|---|
| h_ | ulong | High part. |
| l_ | ulong | Low part. |

### 2.2.66.2 Nonmember Functions

| Function | Description |
|---|---|
| divmod(System.uhuge, System.uhuge) | Computes quotient and remainder of division of two 128-bit values. |
| divmod(System.uhuge, uint) | Computes quotient and remainder of division of a 128-bit value divided by a 32-bit value. |
| mul(System.uhuge, System.uhuge) | Computes the 256-bit product of two 128-bit values. |
| operator%(System.uhuge, System.uhuge) | Computes the remainder of division of two 128-bit values. |
| operator&(System.uhuge, System.uhuge) | Bitwise-AND operation of two 128-bit values. |
| operator*(System.uhuge, System.uhuge) | Computes the product of two 128-bit values. |
| operator+(System.uhuge, System.uhuge) | Computes the sum of two 128-bit values. |
| operator-(System.uhuge, System.uhuge) | Subtracts a 128-bit value from 128-bit value. |
| operator/(System.uhuge, System.uhuge) | Division of two 128-bit values. |
| operator<(System.uhuge, System.uhuge) | Compares two 128-bit values for less than relationship. |
| operator<<(System.uhuge, System.uhuge) | Shifts the given **uhuge** value to the given number of bit positions left and returns it. |
| operator==(System.uhuge, System.uhuge) | Compares two 128-bit values for equality. |
| operator>>(System.uhuge, System.uhuge) | Shifts the given **uhuge** value to given bit positions right and returns it. |

| | |
|---|---|
| operatorˆ(System.uhuge, System.uhuge) | Computes the bitwise-XOR operation of two 128-bit values. |
| operator—(System.uhuge, System.uhuge) | Computes bitwise-OR of two 128-bit values. |
| operator∼(System.uhuge) | Computes bitwise complement of a 128-bit value. |

### divmod(System.uhuge, System.uhuge) Function

Computes quotient and remainder of division of two 128-bit values.

#### Syntax

```
public System.Pair<System.uhuge, System.uhuge> divmod(System.uhuge left, System.uhuge
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | Dividend. |
| right | System.uhuge | Divisor. |

#### Returns

System.Pair<System.uhuge, System.uhuge>

Returns a pair in which the first value is the quotient and the second value is the remainder of $left/right$.

### divmod(System.uhuge, uint) Function

Computes quotient and remainder of division of a 128-bit value divided by a 32-bit value.

#### Syntax

```
public System.Pair<System.uhuge, uint> divmod(System.uhuge left, uint right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | Dividend. |
| right | uint | Divisor. |

#### Returns

System.Pair<System.uhuge, uint>

Returns a pair in which the first value is the quotient and the second value is the remainder of $left/right$.

### mul(System.uhuge, System.uhuge) Function

Computes the 256-bit product of two 128-bit values.

#### Syntax

```
public System.Pair<System.uhuge, System.uhuge> mul(System.uhuge left, System.uhuge
right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

System.Pair<System.uhuge, System.uhuge>

Returns a pair in which the first value is the high part and the second value is the low part of of $left \times right$.

### operator%(System.uhuge, System.uhuge) Function

Computes the remainder of division of two 128-bit values.

#### Syntax

```
public System.uhuge operator%(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | Dividend. |
| right | System.uhuge | Divisor. |

#### Returns

System.uhuge

Returns $left\%right$.

### operator&(System.uhuge, System.uhuge) Function

Bitwise-AND operation of two 128-bit values.

#### Syntax

```
public System.uhuge operator&(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

System.uhuge

Returns bitwise AND of $left$ and $right$.

### operator*(System.uhuge, System.uhuge) Function

Computes the product of two 128-bit values.

#### Syntax

```
public System.uhuge operator*(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | Ths first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

System.uhuge

Returns $left \times right$.

### operator+(System.uhuge, System.uhuge) Function

Computes the sum of two 128-bit values.

#### Syntax

```
public System.uhuge operator+(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

System.uhuge

Returns $left + right$.

### operator-(System.uhuge, System.uhuge) Function

Subtracts a 128-bit value from 128-bit value.

#### Syntax

```
public System.uhuge operator-(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

System.uhuge

Returns $left - right$.

### operator/(System.uhuge, System.uhuge) Function

Division of two 128-bit values.

#### Syntax

```
public System.uhuge operator/(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | Dividend. |
| right | System.uhuge | Divisor. |

#### Returns

System.uhuge

Returns $left$ divided by $right$.

### operator<(System.uhuge, System.uhuge) Function

Compares two 128-bit values for less than relationship.

#### Syntax

```
public bool operator<(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

bool

Returns true, if the first 128-bit value is less than the second 128-bit value, false otherwise.

### operator<<(System.uhuge, System.uhuge) Function

Shifts the given uhuge value to the given number of bit positions left and returns it.

#### Syntax

```
public System.uhuge operator<<(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The uhuge value to shift. |
| right | System.uhuge | The number of bit positions to shift. |

#### Returns

System.uhuge

Returns an uhuge value shifted given number of bit positions left.

### operator==(System.uhuge, System.uhuge) Function

Compares two 128-bit values for equality.

#### Syntax

```
public bool operator==(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

bool

Returns true, if the first 128-bit value is equal to the second 128-bit value, false otherwise.

### operator>>(System.uhuge, System.uhuge) Function

Shifts the given uhuge value to given bit positions right and returns it.

#### Syntax

```
public System.uhuge operator>>(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | An uhuge value to shift. |
| right | System.uhuge | Number of bit positions to shift. |

#### Returns

System.uhuge

Returns the given uhuge value shifted given number of bit positions right.

### operator^(System.uhuge, System.uhuge) Function

Computes the bitwise-XOR operation of two 128-bit values.

#### Syntax

```
public System.uhuge operator^(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

System.uhuge

Returns $left$ XOR $right$.

### operator|(System.uhuge, System.uhuge) Function

Computes bitwise-OR of two 128-bit values.

#### Syntax

```
public System.uhuge operator|(System.uhuge left, System.uhuge right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.uhuge | The first 128-bit value. |
| right | System.uhuge | The second 128-bit value. |

#### Returns

System.uhuge

Returns bitwise-OR of $left$ and $right$.

### operator~(System.uhuge) Function

Computes bitwise complement of a 128-bit value.

#### Syntax

```
public System.uhuge operator~(System.uhuge x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | System.uhuge | A 128-bit value. |

#### Returns

System.uhuge

Returns bitwise complement of $x$.

## 2.3 Type Definitions

| Name | Type | Description |
|---|---|---|
| string | System.String<char> | An alias for UTF-8 encoded string. |
| ustring | System.String<uchar> | An alias for UTF-32 encoded string. |
| wstring | System.String<wchar> | An alias for UTF-16 encoded string. |

## 2.4 Functions

| Function | Description |
|---|---|
| Abs(const T&) | Returns the absolute value of the argument. |
| Accumulate(I, I, T, Op) | Accumulates a sequence with respect to a binary operation. |
| BackInserter(C&) | Returns a BackInsertIterator<C> for a back insertion sequence. |
| CaptureCurrentException() | This function can be used in exception handlers (catch clauses) to capture the current exception with full type information to an ExceptionPtr. This is useful for example in multithreaded programs where exception thrown inside a thread function can be transferred to the main thread and rethrown using RethrowException(System.-ExceptionPtr&) function from there. |
| Copy(I, I, O) | Copies a sequence. |
| CopyBackward(I, I, O) | Copies a source sequence to a target sequence starting from the end of the source sequence. |
| Count(I, I, P) | Counts the number of elements in a sequence that satisfy a predicate. |
| Count(I, I, const T&) | Counts the number of elements in a sequence that are equal to the given value. |
| CurrentDate() | Returns current date as Date value. |
| Distance(I, I) | Returns the distance between two random access iterators. |
| Distance(I, I) | Returns the distance between two forward iterators. |
| EnableSharedFromThis(System.-ShareableFromThis<T>*, U*, const System.-SharedCount<U>&) | A function that enables the *shared from this* idiom. Implementation detail. |
| EnableSharedFromThis(void*, void*, const System.SharedCount<T>&) | An empty function that catches classes that are not shareable from this. Implementation detail. |
| Equal(I1, I1, I2, I2) | Compares two sequences for equality. |
| Equal(I1, I1, I2, I2, R) | Compares two sequences for equality using the given equality relation. |

| | |
|---|---|
| EqualRange(I, I, const T&) | Returns a pair of iterators that form a range of values equal to the given value in a sorted sequence. |
| EqualRange(I, I, const T&, R) | Returns a pair of iterators that form a range of values equal to the given value in a sorted sequence. Uses the given ordering relation to infer equality. |
| Execute(const System.String<char>&) | Executes the given command string by passing to the command interpreter (or shell). |
| Execute(const System.String<char>&, bool) | Executes the given command string by passing to the command interpreter (or shell). |
| Execute(const System.String<char>&, const System.Collections.List<System.Pair<int, System.String<char>>>&) | Executes the given command string by passing to the command interpreter (or shell). Redirects given file descriptors to given files. |
| Execute(const System.String<char>&, const System.Collections.List<System.Pair<int, System.String<char>>>&, bool) | Executes the given command string by passing to the command interpreter (or shell). Redirects given file descriptors to given files. |
| Execute(const System.String<char>&, int, const System.String<char>&) | Executes the given command string by passing to the command interpreter (or shell). Redirects given file handle to given file. |
| Execute(const System.String<char>&, int, const System.String<char>&, bool) | Executes the given command string by passing to the command interpreter (or shell). Redirects given file handle to given file. |
| Factorial(U) | Returns a factorial of the argument. |
| Fill(I, I, const T&) | Fills a sequence with the given value. |
| Find(I, I, P) | Searches the first occurrence of a value from a sequence that matches a predicate. |
| Find(I, I, const T&) | Searches a value from a sequence. |
| ForEach(I, I, F) | Applies a function object for each element of a sequence. |
| FrontInserter(C&) | Returns a FrontInsertIterator<C> for a front insert sequence. |
| Gcd(T, T) | Returns the greatest common divisor of two values. |

| | |
|---|---|
| HexChar(byte) | Returns hexadecimal character representation of a four-bit value. |
| IdentityElement(System.Multiplies<T>) | Returns the identity element of multiplication, that is T(1). |
| IdentityElement(System.Plus<T>) | Returns the identity element of addition, that is: T(0). |
| Inserter(C&, I) | Returns an InsertIterator<C> for an insertion sequence and its iterator. |
| InsertionSort(I, I) | Sorts a sequence of values using insertion sort algorithm. |
| InsertionSort(I, I, R) | Sorts a sequence of values using insertion sort algorithm and given ordering relation. |
| IsAlpha(char) | Returns true if the given character is an alphabetic character, false otherwise. |
| IsAlphanumeric(char) | Returns true if the given character is an alphanumeric character, false otherwise. |
| IsControl(char) | Returns true if the given character is a control character, false otherwise. |
| IsDigit(char) | Returns true if the given character is a decimal digit, false otherwise. |
| IsGraphic(char) | Returns true if the given character is a graphical character, false otherwise. |
| IsHexDigit(char) | Returns true if the given character is a hexadecimal digit, false otherwise. |
| IsLower(char) | Returns true if the given character is a lower case letter, false otherwise. |
| IsPrintable(char) | Returns true if the given character is a printable character, false otherwise. |
| IsPunctuation(char) | Returns true if the given character is a punctuation character, false otherwise. |
| IsSpace(char) | Returns true if the given character is a space character, false otherwise. |

| | |
|---|---|
| IsUpper(char) | Returns true if the given character is an upper case letter, false otherwise. |
| LastComponentsEqual(const System.String<-CharT>&, const System.String<CharT>&, CharT) | Returns true, if last components of two strings are equal, false otherwise. |
| LexicographicalCompare(I1, I1, I2, I2) | Returns true if the first sequence comes lexicographically before the second sequence, false otherwise. |
| LexicographicalCompare(I1, I1, I2, I2, R) | Returns true if the first sequence comes lexicographically before the second sequence according to the given ordering relation, false otherwise. |
| LowerBound(I, I, const T&) | Finds a position of the first element in a sorted sequence that is greater than or equal to the given value. |
| LowerBound(I, I, const T&, R) | Finds a position of the first element in a sorted sequence that is greater than or equal to the given value according to the given ordering relation. |
| MakePair(const T&, const U&) | Returns a pair composed of the given values. |
| Max(const T&, const T&) | Returns the maximum of two values. |
| MaxElement(I, I) | Returns the position of the first occurrence of the largest element in a sequence of elements. |
| MaxElement(I, I, R) | Returns the position of the first occurrence of the largest element according to the given ordering relation in a sequence of elements. |
| MaxValue() | Returns the largest value of an integer type. |
| MaxValue(byte) | Returns the maximum value of **byte**: 255. |
| MaxValue(int) | Returns the maximum value of **int**: 2147483647. |
| MaxValue(long) | Returns the maximum value of **long**: 9223372036854775807. |
| MaxValue(sbyte) | Returns the maximum value of **sbyte**: 127. |
| MaxValue(short) | Returns the maximum value of **short**: 32767. |
| MaxValue(uint) | Returns the maximum value of **uint**: 4294967295. |

| | |
|---|---|
| MaxValue(ulong) | Returns the maximum value of **ulong**: 18446744073709551615. |
| MaxValue(ushort) | Returns the maximum value of **ushort**: 65535. |
| Median(const T&, const T&, const T&) | Returns the median of three values. |
| Median(const T&, const T&, const T&, R) | Returns the median of three values according to the given ordering relation. |
| Min(const T&, const T&) | Returns the minimum of two values. |
| MinElement(I, I) | Returns the position of the first occurrence of the smallest element in a sequence of elements. |
| MinElement(I, I, R) | Returns the position of the first occurrence of the smallest element according to the given ordering relation in a sequence of elements. |
| MinValue() | Returns the smallest value of an integer type. |
| MinValue(byte) | Returns the minimum value of **byte**: 0. |
| MinValue(int) | Returns the minimum value of **int**: -2147483648. |
| MinValue(long) | Returns the minimum value of **long**: -9223372036854775808. |
| MinValue(sbyte) | Returns the minimum value of **sbyte**: -128. |
| MinValue(short) | Returns the minimum value of **short**: -32768. |
| MinValue(uint) | Returns the minimum value of **uint**: 0. |
| MinValue(ulong) | Returns the minimum value of **ulong**: 0. |
| MinValue(ushort) | Returns the minimum value of **ushort**: 0. |
| Move(I, I, O) | Moves a sequence. |
| MoveBackward(I, I, O) | Moves a source sequence to a target sequence starting from the end of the source sequence. |
| Next(I, int) | Returns a random access iterator advanced the specified offset. |
| Next(I, int) | Returns a forward iterator advanced the specified number of steps. |

| | |
|---|---|
| NextPermutation(I, I) | Computes the lexicographically next permutation of a sequence of elements. |
| NextPermutation(I, I, R) | Computes the lexicographically next permutation of a sequence of elements according to the given ordering relation. |
| Now() | Returns current time point value from computer's real time clock. |
| ParseBool(const System.String<char>&) | Parses a Boolean value "true" or "false" from the given string and returns it. |
| ParseBool(const System.String<char>&, bool&) | Parses a Boolean value "true" or "false" from the given string and returns true if the parsing was successful, false if not. |
| ParseDate(const System.String<char>&) | Parses a date from the given string and returns it. |
| ParseDouble(const System.String<char>&) | Parses a **double** value from the given string and returns it. |
| ParseDouble(const System.String<char>&, double&) | Parses a **double** value from the given string and returns true if the parsing was successful, false if not. |
| ParseHex(const System.String<char>&) | Parses a hexadecimal value from a string. |
| ParseHex(const System.String<char>&, System.uhuge&) | Parses a hexadecimal 128-bit value from a string and returns true, if the parsing was successful. |
| ParseHex(const System.String<char>&, ulong&) | Parses a hexadecimal value from a string and returns true if the parsing was successful. |
| ParseHexUHuge(const System.String<char>&) | Parses a 128-bit decimal value from a string. |
| ParseInt(const System.String<char>&) | Parses an **int** from the given string and returns it. |
| ParseInt(const System.String<char>&, int&) | Parses an **int** value from the given string and returns true if the parsing was successful, false if not. |
| ParseLong(const System.String<char>&) | Parses a **long** from the given string and returns it. |
| ParseLong(const System.String<char>&, long&) | Parses a **long** value from the given string and returns true if the parsing was successful, false if not. |

| | |
|---|---|
| ParseUHuge(const System.String<char>&) | Parses a decimal 128-bit value from a string. |
| ParseUHuge(const System.String<char>&, System.uhuge&) | Parses a decimal 128-bit value from a string and returns true, if the parsing was successful. |
| ParseUInt(const System.String<char>&) | Parses an **uint** from the given string and returns it. |
| ParseUInt(const System.String<char>&, uint&) | Parses an **uint** value from the given string and returns true if the parsing was successful, false if not. |
| ParseULong(const System.String<char>&) | Parses an **ulong** from the given string and returns it. |
| ParseULong(const System.String<char>&, ulong&) | Parses an **ulong** value from the given string and returns true if the parsing was successful, false if not. |
| PrevPermutation(I, I) | Computes the lexicographically previous permutation of a sequence of elements. |
| PrevPermutation(I, I, R) | Computes the lexicographically previous permutation according to the given ordering relation of a sequence of elements. |
| PtrCast(const System.SharedPtr<T>&) | Casts a shared pointer. |
| Rand() | Returns a pseudorandom number generated by the Mersenne Twister pseudorandom number generator MT. |
| RandomNumber(uint) | Returns a pseudorandom number in range 0..n - 1 inclusive. |
| RandomShuffle(I, I) | Computes a random permutation of a random access sequence. |
| Remove(I, I, P) | Removes values satisfying a given predicate from a sequence by moving them to the end of the sequence. Returns an iterator pointing to the beginning of removed values. |
| Remove(I, I, const T&) | Removes the values equal to the given value from the given sequence of values by moving them to the end of the sequence. Returns an iterator pointing to the beginning of removed values. |

| | |
|---|---|
| RemoveCopy(I, I, O, P) | Copies elements from given range to a destination range excluding elements satisfying a given predicate. Returns an iterator pointing one past the end of the destination range. |
| RemoveCopy(I, I, O, const T&) | Copies elements from given range to a destination range excluding elements that are equal to given value. Returns an iterator pointing one past the end of the destination range. |
| RethrowException(System.ExceptionPtr&) | Rethrows an exception captured to an ExceptionPtr. |
| Reverse(I, I) | Reverses a sequence. |
| Reverse(I, I) | Reverses a sequence. |
| ReverseUntil(I, I, I) | Reverses a sequence until the given middle iterator is hit. |
| Rotate(I, I, I) | Rotates a sequence with respect to a middle iterator and returns an iterator pointing to the new middle. |
| Rvalue(T&&) | Converts an argument to an rvalue so that it can be moved. |
| Select_0_2(const T&, const T&, R) | Returns the smaller of two values according to the given ordering relation. |
| Select_0_3(const T&, const T&, const T&, R) | Returns the smallest of tree values according to the given ordering relation. |
| Select_1_2(const T&, const T&, R) | Returns the larger of two values according to the given ordering relation. |
| Select_1_3(const T&, const T&, const T&, R) | Returns the median of three values according to the given ordering relation. |
| Select_1_3_ab(const T&, const T&, const T&, R) | Returns the median of three values when the first two are in increasing order according to the given ordering relation. |
| Select_2_3(const T&, const T&, const T&, R) | Returns the largest of three values according to the given ordering relation. |

| | |
|---|---|
| Sort(C&) | Sorts the elements of a random access container to increasing order. |
| Sort(C&) | Sorts the elements of a forward container to increasing order. |
| Sort(C&, R) | Sorts the elements of a random access container to order according to the given ordering relation. |
| Sort(C&, R) | Sorts the elements of a forward container to order according to the given ordering relation. |
| Sort(I, I) | Sorts the elements of a sequence to increasing order. |
| Sort(I, I, R) | Sorts the elements of a sequence to order according to the given ordering relation. |
| Swap(T&, T&) | Exchanges two values. |
| ThrowConversionException(const System.-String<char>&) | Throws ConversionException with the given error message. |
| ToHexString(System.uhuge) | Returns a 128-bit value converted to hexadecimal representation. |
| ToHexString(U) | Converts an unsigned integer value to hexadecimal string representation. |
| ToHexString(byte) | Converts a **byte** to hexadecimal string representation. |
| ToHexString(uint) | Converts an **uint** to hexadecimal string representation. |
| ToHexString(ulong) | Converts an **ulong** to hexadecimal string representation. |
| ToHexString(ushort) | Converts an **ushort** to hexadecimal string representation. |
| ToLower(const System.String<char>&) | Converts the given UTF-8 string to lower case. |
| ToLower(const System.String<uchar>&) | Converts the given UTF-32 string to lower case. |
| ToLower(const System.String<wchar>&) | Converts the given UTF-16 string to lower case. |

| | |
|---|---|
| ToString(I) | Converts a signed integer value to string representation. |
| ToString(System.Date) | Converts a Date to a string representation. |
| ToString(System.uhuge) | Converts a 128-bit value to string representation. |
| ToString(U) | Converts an unsigned integer value to string representation. |
| ToString(bool) | Converts a Boolean value to string representation. |
| ToString(byte) | Converts a **byte** to string representation. |
| ToString(char) | Converts a character to string representation. |
| ToString(double) | Converts a **double** to string representation. |
| ToString(double, int) | Converts a **double** to string representation using the given maximum number of decimal places. |
| ToString(double, int, int) | Converts a **double** to string representation using the given minimum and maximum number of decimal places. |
| ToString(int) | Converts an **int** to string representation. |
| ToString(long) | Converts a **long** to string representation. |
| ToString(sbyte) | Converts an **sbyte** to string representation. |
| ToString(short) | Converts a **short** to string representation. |
| ToString(uchar) | Converts the given UTF-32 character to a UTF-8 string representation. |
| ToString(uint) | Converts an **uint** to string representation. |
| ToString(ulong) | Converts an **ulong** to string representation. |
| ToString(ushort) | Converts an **ushort** to string representation. |
| ToString(wchar) | Converts the given UTF-16 character to a UTF-8 string representation. |
| ToUpper(const System.String<char>&) | Converts the given UTF-8 string to upper case. |

| | |
|---|---|
| ToUpper(const System.String<uchar>&) | Converts the given UTF-32 string to upper case. |
| ToUpper(const System.String<wchar>&) | Converts the given UTF-16 string to upper case. |
| Transform(I, I, O, F) | Transforms an input sequence to an output sequence using a unary function. |
| Transform(I1, I1, I2, O, F) | Transforms two input sequences to an ouput sequence using a binary function. |
| UpperBound(I, I, const T&) | Finds a position of the first element in a sorted sequence that is greater than the given value. |
| UpperBound(I, I, const T&, R) | Finds a position of the first element in a sorted sequence that is greater than the given value according to the given ordering relation. |
| endl() | Returns EndLine object that represents an end of line character. |

## 2.4.67 Abs(const T&) Function

Returns the absolute value of the argument.

### Syntax

```
public T Abs(const T& x);
```

### Constraint

T is OrderedAdditiveGroup

### Parameters

| Name | Type | Description |
|---|---|---|
| x | const T& | A value. |

### Returns

T

if $x < T(0)$ returns $-x$, else returns $x$.

#### 2.4.67.1 Example

```
using System;

//  Writes:
//  0
//  5
//  10

void main()
{
    int zero = 0;
    int minusFive = -5;
    int ten = 10;
//  ...
    Console.Out() << Abs(zero) << endl();
    Console.Out() << Abs(minusFive) << endl();
    Console.Out() << Abs(ten) << endl();
}
```

## 2.4.68 Accumulate(I, I, T, Op) Function

Accumulates a sequence with respect to a binary operation.

### Syntax

```
public T Accumulate(I begin, I end, T init, Op op);
```

### Constraint

I is InputIterator and T is Semiregular and Op is BinaryOperation and Op.FirstArgumentType is T and Op.SecondArgumentType is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of a sequence. |
| end | I | An input iterator pointing one past the end of a sequence. |
| init | T | Initial value. |
| op | Op | A binary operation. |

### Returns

T

Returns accumulated result.

### Remarks

When the binary operation is Plus<T> and *init* is zero calculates the sum of a sequence. When the binary operation is Multiplies<T> and *init* is one calculates the product of a sequence.

#### 2.4.68.1 Example

```
using System;
using System.Collections;

//  Writes:
//  6

void main()
{
    List<int> ints;
    ints.Add(1);
    ints.Add(2);
    ints.Add(3);
    int init = 0;
    int sum = Accumulate(ints.CBegin(), ints.CEnd(), init, Plus<int>());
    Console.WriteLine(sum);
}
```

### Implementation

algorithm.cm, page 8

## 2.4.69 BackInserter(C&) Function

Returns a BackInsertIterator<C> for a back insertion sequence.

### Syntax

```
public System.BackInsertIterator<C> BackInserter(C& c);
```

### Constraint

C is BackInsertionSequence

### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | A back insertion sequence. |

### Returns

System.BackInsertIterator<C>

Returns a BackInsertIterator<C> for a back insertion sequence.

### Remarks

A BackInsertIterator<C> is an output iterator that inserts elements to the end of a back insertion sequence.

#### 2.4.69.1 Example

```
using System;
using System.Collections;

//  Writes:
//  1, 2, 3

void main()
{
    Set<int> s;
    s.Insert(2);
    s.Insert(3);
    s.Insert(1);
//  Set is a sorted container, so it contains now 1, 2, 3...

    List<int> list; // list is a model of a back insertion sequence

//  Copy ints from s to the end of list using BackInsertIterator...
    Copy(s.CBegin(), s.CEnd(), BackInserter(list));

//  ForwardContainers containing ints can be put to OutputStream...
    Console.Out() << list << endl();
}
```

## 2.4.70 CaptureCurrentException() Function

This function can be used in exception handlers (catch clauses) to capture the current exception with full type information to an ExceptionPtr. This is useful for example in multithreaded programs where exception thrown inside a thread function can be transferred to the main thread and rethrown using RethrowException(System.ExceptionPtr&) function from there.

### Syntax

```
public System.ExceptionPtr CaptureCurrentException();
```

### Returns

System.ExceptionPtr

Returns ExceptionPtr

### 2.4.70.1 Example

```
using System;

// This program writes the following message to the standard error stream:
// FooException at 'C:/Temp/exptr/System.ExceptionPtr.cm' line 32:
// exception from thread
// call stack:
// 1> function 'foo()' file C:/Temp/exptr/System.ExceptionPtr.cm line 32
// 0> function 'ThreadFunction(void*)' file C:/Temp/exptr/System.
    ExceptionPtr.cm line 40

public class FooException : Exception
{
    public FooException(const string& message_) : base(message_)
    {
    }
}

public class ThreadData
{
    public void SetException(ExceptionPtr&& exPtr_)
    {
        exPtr = exPtr_;
    }
    public ExceptionPtr GetException() const
    {
        return Rvalue(exPtr);
    }
    private ExceptionPtr exPtr;
}

void foo()
{
    throw FooException("exception from thread");
}

void ThreadFunction(void* data)
{
    ThreadData* threadData = cast<ThreadData*>(data);
```

```
    try
    {
        foo();
    }
    catch (const Exception& ex)
    {
        ExceptionPtr exPtr = CaptureCurrentException();
        threadData->SetException(Rvalue(exPtr));
    }
}

void main()
{
    try
    {
        ThreadData threadData;
        System.Threading.Thread thread(System.Threading.ThreadFun(
            ThreadFunction), &threadData);
        thread.Join();
        ExceptionPtr exPtr = threadData.GetException();
        if (exPtr.HasException())
        {
            RethrowException(exPtr);
        }
    }
    catch (const Exception& ex)
    {
        Console.Error() << ex.ToString() << endl();
    }
}
```

## 2.4.71 Copy(I, I, O) Function

Copies a sequence.

### Syntax

```
public O Copy(I begin, I end, O to);
```

### Constraint

I is InputIterator and O is OutputIterator and CopyAssignable<O.ValueType, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of a the source sequence. |
| end | I | An input iterator pointing one past the end of a source sequence. |
| to | O | An output iterator pointing to the beginning of the target sequence. |

### Returns

O

Returns an output iterator pointing one past the end of the copied sequence.

### Remarks

The source and target sequences may overlap, but then the iterator pointing to the beginning of the target sequence must point to an object coming before the object pointed by the beginning iterator of the source sequence.

#### 2.4.71.1 Example

```
using System;
using System.Collections;
using System.IO;

//  Writes:
//  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
//  5, 6, 7, 8, 9, 0, 1, 2, 3, 4

void main()
{
    List<int> source;
    int n = 10;
    for (int i = 0; i < n; ++i)
    {
        source.Add(i);
    }
```

```
    Console.Out() << source << endl();
    List<int> target;
    Copy(source.CBegin() + n / 2, source.CEnd(), BackInserter(target));
    Copy(source.CBegin(), source.CBegin() + n / 2, BackInserter(target));
    Console.Out() << target << endl();
}
```

## Implementation

algorithm.cm, page 2

## 2.4.72 CopyBackward(I, I, O) Function

Copies a source sequence to a target sequence starting from the end of the source sequence.

### Syntax

```
public O CopyBackward(I begin, I end, O to);
```

### Constraint

I is BidirectionalIterator and O is BidirectionalIterator and CopyAssignable<O.ValueType, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A bidirectional iterator pointing to the beginning of the source sequence. |
| end | I | A bidirectional iterator pointing one past the end of the source sequence. |
| to | O | A bidirectional iterator pointing one past the end of the target sequence. |

### Returns

O

Returns a birectional iterator pointing to the beginning of the target sequence.

### Remarks

The source and target sequences may overlap, but then the end iterator of the target sequence must point to an element coming after the element pointed by end iterator of the source sequence.

#### 2.4.72.1 Example

```
using System;
using System.Collections;
using System.IO;

//  Writes:
//  0, 1, 2, 3

void main()
{
    List<int> list;
    list.Add(0);
    list.Add(2);
    list.Add(3);
    list.Add(3);
    CopyBackward(list.CBegin() + 1, list.CEnd() - 1, list.End());
```

```
    list[1] = 1;
    Console.Out() << list << endl();
}
```

## Implementation

algorithm.cm, page 2

## 2.4.73 Count(I, I, P) Function

Counts the number of elements in a sequence that satisfy a predicate.

### Syntax

```
public int Count(I begin, I end, P p);
```

### Constraint

I is InputIterator and P is UnaryPredicate and P.ArgumentType is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of a sequence. |
| end | I | An input iterator pointing one past the end of a sequence. |
| p | P | A unary predicate. |

### Returns

int

Returns the number of elements that satisfy *p*.

#### 2.4.73.1 Example

```
using System;
using System.Concepts;
using System.Collections;

//  Writes:
//  3

public class Even<I>: UnaryPred<I> where I is SignedInteger
{
    public inline nothrow bool operator()(I n) const
    {
        return n % I(2) == I(0);
    }
}

void main()
{
    List<int> list;
    list.Add(0);
    list.Add(2);
    list.Add(3);
    list.Add(4);
    list.Add(5);
    Console.Out() << Count(list.CBegin(), list.CEnd(), Even<int>()) << endl
        ();
}
```

### Implementation

algorithm.cm, page 7

## 2.4.74 Count(I, I, const T&) Function

Counts the number of elements in a sequence that are equal to the given value.

### Syntax

```
public int Count(I begin, I end, const T& value);
```

### Constraint

I is InputIterator and T is Semiregular and EqualityComparable<T, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of a sequence. |
| end | I | An input iterator pointing one past the end of a sequence. |
| value | const T& | A value. |

### Returns

int

Returns the number of elements equal to *value*.

#### 2.4.74.1 Example

```
using System;
using System.Collections;

//  Writes:
//  3

void main()
{
    List<string> animals;
    animals.Add("cat");
    animals.Add("cat");
    animals.Add("lion");
    animals.Add("dog");
    animals.Add("mouse");
    animals.Add("lion");
    animals.Add("moose");
    animals.Add("lion");
    animals.Add("bear");
    animals.Add("dog");
    string lion("lion");
    Console.Out() << Count(animals.CBegin(), animals.CEnd(), lion) << endl()
        ;
}
```

### Implementation

algorithm.cm, page 7

### 2.4.75 CurrentDate() Function

Returns current date as Date value.

#### Syntax

```
public System.Date CurrentDate();
```

#### Returns

System.Date

Returns current date as Date value.

## 2.4.76 Distance(I, I) Function

Returns the distance between two random access iterators.

### Syntax

```
public int Distance(I first, I last);
```

### Constraint

I is RandomAccessIterator

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | The first random access iterator. |
| last | I | The second random access iterator. |

### Returns

int

Returns $last - first$.

#### 2.4.76.1 Example

```
using System;
using System.Collections;

//  Writes:
//  4

void main()
{
    List<int> list;
    for (int i = 0; i < 10; ++i)
    {
        list.Add(i + 3);
    }
    List<int>.ConstIterator it = Find(list.CBegin(), list.CEnd(), 7);
    int indexOfSeven = Distance(list.CBegin(), it);
    Console.Out() << indexOfSeven << endl();
}
```

### Implementation

algorithm.cm, page 3

## 2.4.77 Distance(I, I) Function

Returns the distance between two forward iterators.

### Syntax

```
public int Distance(I first, I last);
```

### Constraint

I is ForwardIterator

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | The first forward iterator. |
| last | I | The second forward iterator reachable from the first forward iterator. |

### Returns

int

Returns the number of steps that the *first* iterator must be incremented to reach the *last* iterator.

### Remarks

The *last* iterator must be reachable from the *first* iterator.

### Implementation

algorithm.cm, page 3

### 2.4.78 EnableSharedFromThis(System.ShareableFromThis<T>*, U*, const System.SharedCount<U>&) Function

A function that enables the *shared from this* idiom. Implementation detail.

#### Syntax

```
public void EnableSharedFromThis(System.ShareableFromThis<T>* left, U* right, const
System.SharedCount<U>& count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.ShareableFromThis<T>* | A pointer to a class derived from ShareableFromThis<T>. |
| right | U* | A pointer to class derived from ShareableFromThis<T>. |
| count | const System.SharedCount<U>& | A shared count. |

### 2.4.79 EnableSharedFromThis(void*, void*, const System.SharedCount Function

An empty function that catches classes that are not shareable from this. Implementation detail.

#### Syntax

```
public void EnableSharedFromThis(void* __parameter0, void* __parameter1, const System.SharedCou
__parameter2);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | void* | Pointer to any object. |
| __parameter1 | void* | Pointer to any object. |
| __parameter2 | const System.SharedCount<T>& | A shared count. |

## 2.4.80 Equal(I1, I1, I2, I2) Function

Compares two sequences for equality.

### Syntax

```
public bool Equal(I1 first1, I1 last1, I2 first2, I2 last2);
```

### Constraint

I1 is InputIterator and I2 is InputIterator and EqualityComparable<I1.ValueType, I2.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| first1 | I1 | An input iterator pointing to the beginning of the first sequence. |
| last1 | I1 | An input iterator pointer one past the end of the first sequence. |
| first2 | I2 | An input iterator pointing to the beginning of the second sequence. |
| last2 | I2 | An input iterator pointing one past the end of the second sequence. |

### Returns

bool

Returns true if the first sequence is equal to the second sequence, false otherwise.

### Remarks

Two sequences are equal if they contain the same number of elements that compare pairwise equal. Uses the EqualTo2<T, U> binary predicate to compare the elements of the sequences.

#### 2.4.80.1 Example

```
using System;
using System.Collections;

//  Writes:
//  set: 41, 6334, 11478, 15724, 18467, 19169, 24464, 26500, 26962, 29358
//  list: 41, 18467, 6334, 26500, 19169, 15724, 11478, 29358, 26962, 24464
//  sorted list: 41, 6334, 11478, 15724, 18467, 19169, 24464, 26500, 26962,
    29358
//  this was expected

void main()
{
    Set<int> set;
```

```
    List<int> list;
    int n = 10;
    for (int i = 0; i < n; ++i)
    {
        int r = rand();
        set.Insert(r);
        list.Add(r);
    }
    Console.Out() << "set: " << set << endl();
    Console.Out() << "list: " << list << endl();
    Sort(list);
    Console.Out() << "sorted list: " << list << endl();
    if (Equal(list.CBegin(), list.CEnd(), set.CBegin(), set.CEnd()))
    {
        Console.Out() << "this was expected" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

## Implementation

algorithm.cm, page 13

## 2.4.81 Equal(I1, I1, I2, I2, R) Function

Compares two sequences for equality using the given equality relation.

### Syntax

```
public bool Equal(I1 first1, I1 last1, I2 first2, I2 last2, R r);
```

### Constraint

I1 is InputIterator and I2 is InputIterator and Relation<R, I1.ValueType, I2.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| first1 | I1 | An input iterator pointing to the beginning of the first sequence. |
| last1 | I1 | An input iterator pointing one past the end of the first sequence. |
| first2 | I2 | An input iterator pointing to the beginning of the second sequence. |
| last2 | I2 | An input iterator pointing one past the end of the second sequence. |
| r | R | An equality relation. |

### Returns

bool

Returns true if the first sequence is equal to the second sequence according to the given equality relation.

#### 2.4.81.1 Example

```
using System;
using System.Collections;

//  Writes:
//  true

public class A
{
    public A(): id()
    {
    }
    public A(const string& id_): id(id_)
    {
    }
    public nothrow const string& Id() const
    {
```

```
        return id;
    }
    private string id;
}

public class ALess: Rel<A>
{
    public nothrow inline bool operator()(const A& left, const A& right)
        const
    {
        return left.Id() < right.Id();
    }
}

public class AEq: Rel<A>
{
    public nothrow inline bool operator()(const A& left, const A& right)
        const
    {
        return left.Id() == right.Id();
    }
}

void main()
{
    List<A> list;
    list.Add(A("bar"));
    list.Add(A("baz"));
    list.Add(A("foo"));
    Set<A, ALess> set;
    set.Insert(A("foo"));
    set.Insert(A("bar"));
    set.Insert(A("baz"));
    Console.Out() << Equal(list.CBegin(), list.CEnd(), set.CBegin(), set.
        CEnd(), AEq()) << endl();
}
```

## Implementation

algorithm.cm, page 12

## 2.4.82 EqualRange(I, I, const T&) Function

Returns a pair of iterators that form a range of values equal to the given value in a sorted sequence.

### Syntax

```
public System.Pair<I, I> EqualRange(I first, I last, const T& value);
```

### Constraint

I is ForwardIterator and TotallyOrdered<T, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator that points to the beginning of a sorted sequence. |
| last | I | A forward iterator that points to one past the end of a sorted sequence. |
| value | const T& | A value. |

### Returns

System.Pair<I, I>

Returns a pair of iterators that form a range of values equal to the given value in a sorted sequence.

### Remarks

If the value is not found in the sorted sequence, returns a pair of iterators that form an empty range (that is: a range with two equal iterators.) The iterators point to the position where the given value would be if it were in the sorted sequence.

#### 2.4.82.1 Example

```
using System;
using System.Collections;

//  Writes:
//  number 1 occurs 0 times
//  number 2 occurs 1 times
//  number 3 occurs 2 times
//  number 4 occurs 3 times

void main()
{
    List<int> list;
    list.Add(4);
    list.Add(2);
    list.Add(3);
```

```
    list.Add(4);
    list.Add(3);
    list.Add(4);
    Sort(list); // EqualRange needs a sorted sequence
    Pair<List<int>.ConstIterator, List<int>.ConstIterator> p1 = EqualRange(
        list.CBegin(), list.CEnd(), 1);
    Console.Out() << "number " << 1 << " occurs " << Distance(p1.first, p1.
        second) << " times" << endl();
    Pair<List<int>.ConstIterator, List<int>.ConstIterator> p2 = EqualRange(
        list.CBegin(), list.CEnd(), 2);
    Console.Out() << "number " << 2 << " occurs " << Distance(p2.first, p2.
        second) << " times" << endl();
    Pair<List<int>.ConstIterator, List<int>.ConstIterator> p3 = EqualRange(
        list.CBegin(), list.CEnd(), 3);
    Console.Out() << "number " << 3 << " occurs " << Distance(p3.first, p3.
        second) << " times" << endl();
    Pair<List<int>.ConstIterator, List<int>.ConstIterator> p4 = EqualRange(
        list.CBegin(), list.CEnd(), 4);
    Console.Out() << "number " << 4 << " occurs " << Distance(p4.first, p4.
        second) << " times" << endl();
}
```

### 2.4.82.2 Example

```
using System;

//  Writes:
//  sentence 'A bird in the hand is worth two in the bush.' contains 2 'a'
    letters

void main()
{
    string proverb("A bird in the hand is worth two in the bush.");
    string letters = ToLower(proverb);
    Sort(letters.Begin(), letters.End());
    Pair<string.ConstIterator, string.ConstIterator> a = EqualRange(letters.
        CBegin(), letters.CEnd(), 'a');
    Console.Out() << "sentence '" << proverb << "' contains " << Distance(a.
        first, a.second) << " 'a' letters" << endl();
}
```

## Implementation

algorithm.cm, page 5

## 2.4.83 EqualRange(I, I, const T&, R) Function

Returns a pair of iterators that form a range of values equal to the given value in a sorted sequence. Uses the given ordering relation to infer equality.

### Syntax

```
public System.Pair<I, I> EqualRange(I first, I last, const T& value, R r);
```

### Constraint

I is ForwardIterator and T is I.ValueType and R is Relation and R.Domain is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator that points to the beginning of a sorted sequence. |
| last | I | A forward iterator that points to one past the end of a sorted sequence. |
| value | const T& | A value. |
| r | R | An ordering relation. |

### Returns

System.Pair<I, I>

Returns a pair of iterators that form a range of values equal to the given value in a sorted sequence.

### Remarks

If the value is not found in the sorted sequence, returns a pair of iterators that form an empty range (that is: a range with two equal iterators.) The iterators point to the position where the given value would be if it were in the sorted sequence.

#### 2.4.83.1 Example

```
using System;
using System.Collections;

//  Writes:
//  A(1) occurs 0 times
//  A(2) occurs 1 times
//  A(3) occurs 2 times
//  A(4) occurs 3 times

public class A
{
    public A(): id(0)
    {
```

```
    }
    public A(int id_): id(id_)
    {
    }
    public inline nothrow int Id() const
    {
        return id;
    }
    private int id;
}

public class ALess: Rel<A>
{
    public inline nothrow bool operator()(const A& left, const A& right)
    {
        return left.Id() < right.Id();
    }
}

void main()
{
    List<A> list;
    list.Add(A(4));
    list.Add(A(2));
    list.Add(A(3));
    list.Add(A(4));
    list.Add(A(3));
    list.Add(A(4));
    Sort(list, ALess()); // EqualRange needs a sorted sequence
    Pair<List<A>.ConstIterator, List<A>.ConstIterator> p1 = EqualRange(list.
        CBegin(), list.CEnd(), A(1), ALess());
    Console.Out() << "A(" << 1 << ") occurs " << Distance(p1.first, p1.
        second) << " times" << endl();
    Pair<List<A>.ConstIterator, List<A>.ConstIterator> p2 = EqualRange(list.
        CBegin(), list.CEnd(), A(2), ALess());
    Console.Out() << "A(" << 2 << ") occurs " << Distance(p2.first, p2.
        second) << " times" << endl();
    Pair<List<A>.ConstIterator, List<A>.ConstIterator> p3 = EqualRange(list.
        CBegin(), list.CEnd(), A(3), ALess());
    Console.Out() << "A(" << 3 << ") occurs " << Distance(p3.first, p3.
        second) << " times" << endl();
    Pair<List<A>.ConstIterator, List<A>.ConstIterator> p4 = EqualRange(list.
        CBegin(), list.CEnd(), A(4), ALess());
    Console.Out() << "A(" << 4 << ") occurs " << Distance(p4.first, p4.
        second) << " times" << endl();
}
```

## Implementation

algorithm.cm, page 6

### 2.4.84 Execute(const System.String<char>&) Function

Executes the given command string by passing to the command interpreter (or shell).

#### Syntax

```
public void Execute(const System.String<char>& command);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| command | const System.String<char>& | A command to execute. |

#### Remarks

Throws an exception if the command returns a non-zero return value.

### 2.4.85 Execute(const System.String<char>&, bool) Function

Executes the given command string by passing to the command interpreter (or shell).

**Syntax**

```
public void Execute(const System.String<char>& command, bool ignoreReturnValue);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| command | const System.String<char>& | A command to execute. |
| ignoreReturnValue | bool | If true, ignores the value returned by the command, otherwise throws an exception if the return value is non-zero. |

### 2.4.86 Execute(const System.String<char>&, const System.Collections System.String<char>>>&) Function

Executes the given command string by passing to the command interpreter (or shell). Redirects given file descriptors to given files.

#### Syntax

```
public void Execute(const System.String<char>& command, const System.Collections.List<System.Pa
System.String<char>>>& redirections);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| command | const System.String<char>& | A command to execute. |
| redirections | const System.Collections.List<System.Pair<-int, System.String<char>>>& | A list of file descriptor filename pairs. |

### 2.4.87 Execute(const System.String<char>&, const System.Collections System.String<char>>>&, bool) Function

Executes the given command string by passing to the command interpreter (or shell). Redirects given file descriptors to given files.

#### Syntax

```
public void Execute(const System.String<char>& command, const System.Collections.List<System.Pa
System.String<char>>>& redirections, bool ignoreReturnValue);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| command | const System.String<char>& | A command to execute. |
| redirections | const System.Collections.List<-System.Pair<int, System.String<-char>>>& | A list of file descriptor filename pairs. |
| ignoreReturnValue | bool | If true ignores the value returned by the command, otherwise throws an exception if the return value is non-zero. |

### 2.4.88 Execute(const System.String<char>&, int, const System.String< Function

Executes the given command string by passing to the command interpreter (or shell). Redirects given file handle to given file.

#### Syntax

```
public void Execute(const System.String<char>& command, int redirectFd, const System.String<char
toFile);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| command | const System.String<char>& | A command to execute. |
| redirectFd | int | A file descriptor to redirect. |
| toFile | const System.String<char>& | The name of the file to redirect to. |

### 2.4.89 Execute(const System.String<char>&, int, const System.String< bool) Function

Executes the given command string by passing to the command interpreter (or shell). Redirects given file handle to given file.

#### Syntax

```
public void Execute(const System.String<char>& command, int redirectFd, const System.String<cha
toFile, bool ignoreReturnValue);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| command | const System.String<char>& | A command to execute. |
| redirectFd | int | A file descriptor to redirect. |
| toFile | const System.String<char>& | The name of the file to redirect to. |
| ignoreReturnValue | bool | If true, ignores the value returned by the command, otherwise throws an exception if the return value is non-zero. |

## 2.4.90 Factorial(U) Function

Returns a factorial of the argument.

### Syntax

```
public U Factorial(U n);
```

### Constraint

U is UnsignedInteger

### Parameters

| Name | Type | Description |
|---|---|---|
| n | U | An unsigned integer value. |

### Returns

U

Returns $n!$.

#### 2.4.90.1 Example

```
using System;

//  Writes:
//  120

void main()
{
    uint x = 5u;
    uint f = Factorial(x);
    Console.Out() << f << endl();
}
```

### Implementation

algorithm.cm, page 15

## 2.4.91 Fill(I, I, const T&) Function

Fills a sequence with the given value.

### Syntax

```
public void Fill(I begin, I end, const T& value);
```

### Constraint

T is Semiregular and I is ForwardIterator and I.ValueType is T

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An iterator pointing to the beginning of a sequence. |
| end | I | A iterator pointing one past the end of a sequence. |
| value | const T& | A value. |

## 2.4.92 Find(I, I, P) Function

Searches the first occurrence of a value from a sequence that matches a predicate.

### Syntax

```
public I Find(I begin, I end, P p);
```

### Constraint

I is InputIterator and P is UnaryPredicate and P.ArgumentType is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of a sequence. |
| end | I | An input iterator pointing one past the end of a sequence. |
| p | P | A unary predicate. |

### Returns

I

Returns an iterator pointing to the found value, or *end* if no value satisfied *p*.

#### 2.4.92.1 Example

```
using System;
using System.Collections;

//  Writes:
//  index of first odd int is 3

public class Odd<I>: UnaryPred<I> where I is SignedInteger
{
    public nothrow inline bool operator()(I n)
    {
        return n % I(2) == I(1);
    }
}

void main()
{
    List<int> list;
    list.Add(2);
    list.Add(4);
    list.Add(6);
    list.Add(3);
    list.Add(8);
    list.Add(9);
    List<int>.ConstIterator p = Find(list.CBegin(), list.CEnd(), Odd<int>())
        ;
    if (p != list.CEnd())
```

```
    {
        Console.Out() << "index of first odd int is " << Distance(list.
            CBegin(), p) << endl();
    }
    else
    {
        Console.Out() << "no odd int found" << endl();
    }
}
```

## Implementation

algorithm.cm, page 7

## 2.4.93 Find(I, I, const T&) Function

Searches a value from a sequence.

### Syntax

```
public I Find(I begin, I end, const T& value);
```

### Constraint

I is InputIterator and T is Semiregular and EqualityComparable<T, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of a sequence. |
| end | I | An input iterator pointing one past the end of a sequence. |
| value | const T& | A value to search. |

### Returns

I

Returns an iterator pointing to the found value, or *end* if no equal value found.

#### 2.4.93.1 Example

```
using System;
using System.Collections;

//  Writes:
//  A(2) found in index 3

public class A
{
    public A(): id(0)
    {
    }
    public A(int id_): id(id_)
    {
    }
    public nothrow inline int Id() const
    {
        return id;
    }
    private int id;
}

public nothrow inline bool operator==(const A& left, const A& right)
{
    return left.Id() == right.Id();
```

```
}

void main()
{
    List<A> list;
    list.Add(A(3));
    list.Add(A(1));
    list.Add(A(4));
    list.Add(A(2));
    list.Add(A(3));
    list.Add(A(5));
    list.Add(A(2));
    List<A>.ConstIterator i = Find(list.CBegin(), list.CEnd(), A(2));
    if (i != list.CEnd())
    {
        Console.Out() << "A(2) found in index " << Distance(list.CBegin(), i
            ) << endl();
    }
    else
    {
        Console.Out() << "A(2) not found" << endl();
    }
}
```

## Implementation

algorithm.cm, page 6

## 2.4.94 ForEach(I, I, F) Function

Applies a function object for each element of a sequence.

### Syntax

```
public F ForEach(I begin, I end, F f);
```

### Constraint

I is InputIterator and F is UnaryFunction and F.ArgumentType is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of a sequence. |
| end | I | An input iterator pointing one past the end of a sequence. |
| f | F | A unary function object. |

### Returns

F

Returns the function object.

#### 2.4.94.1 Example

```
using System;
using System.Collections;

//  Writes:
//  Triangle.Draw()
//  Circle.Draw()
//  Rectangle.Draw()
//  ~Triangle()
//  ~Circle()
//  ~Rectangle()

public abstract class Figure
{
    public virtual ~Figure()
    {
    }
    public abstract void Draw();
}

public typedef SharedPtr<Figure> FigurePtr;

public class Circle: Figure
{
    public override ~Circle()
    {
        Console.WriteLine("~Circle()");
```

```
    }
    public override void Draw()
    {
        Console.Out() << "Circle.Draw()" << endl();
    }
}

public class Rectangle: Figure
{
    public override ~Rectangle()
    {
        Console.WriteLine("~Rectangle()");
    }
    public override void Draw()
    {
        Console.Out() << "Rectangle.Draw()" << endl();
    }
}

public class Triangle: Figure
{
    public override ~Triangle()
    {
        Console.WriteLine("~Triangle()");
    }
    public override void Draw()
    {
        Console.Out() << "Triangle.Draw()" << endl();
    }
}

public class Draw: UnaryFun<FigurePtr, void>
{
    public void operator()(FigurePtr figure) const
    {
        figure->Draw();
    }
}

public void main()
{
    List<FigurePtr> figures;
    figures.Add(FigurePtr(new Triangle()));
    figures.Add(FigurePtr(new Circle()));
    figures.Add(FigurePtr(new Rectangle()));
    ForEach(figures.CBegin(), figures.CEnd(), Draw());
}
```

## Implementation

algorithm.cm, page 8

## 2.4.95 FrontInserter(C&) Function

Returns a FrontInsertIterator<C> for a front insert sequence.

### Syntax

```
public System.FrontInsertIterator<C> FrontInserter(C& c);
```

### Constraint

C is FrontInsertionSequence

### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | A front insertion sequence. |

### Returns

System.FrontInsertIterator<C>

Returns a FrontInsertIterator<C> for a front insertion sequence.

### Remarks

A FrontInsertIterator<C> is an output iterator that inserts elements to the front of a front insertion sequence.

#### 2.4.95.1 Example

```
using System;
using System.Collections;

//  Writes:
//  list: 3, 2, 1

void main()
{
    Set<int> s;
    s.Insert(2);
    s.Insert(1);
    s.Insert(3);
//  Set is a sorted container, so it contains now 1, 2, 3..

    List<int> list; // list is a model of a front insertion sequence

//  Copy each element in the set to the front of list using
    FrontInsertIterator...
    Copy(s.CBegin(), s.CEnd(), FrontInserter(list));

//  ForwardContainers containing ints can be put to OutputStream...
    Console.Out() << "list: " << list << endl();
}
```

## 2.4.96 Gcd(T, T) Function

Returns the greatest common divisor of two values.

### Syntax

```
public T Gcd(T a, T b);
```

### Constraint

T is EuclideanSemiring

### Parameters

| Name | Type | Description |
|---|---|---|
| a | T | The first value. |
| b | T | The second value. |

### Returns

T

Returns $gcd(a, b)$.

#### 2.4.96.1 Example

```
using System;

//  Writes:
//  4

void main()
{
    Console.WriteLine(Gcd(12, 8));
}
```

### Implementation

algorithm.cm, page 15

## 2.4.97 HexChar(byte) Function

Returns hexadecimal character representation of a four-bit value.

### Syntax

```
public char HexChar(byte nibble);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| nibble | byte | A four bit value. |

### Returns

char

Returns hexadecimal character representation of a four-bit value.

## 2.4.98 IdentityElement(System.Multiplies<T>) Function

Returns the identity element of multiplication, that is T(1).

### Syntax

```
public T IdentityElement(System.Multiplies<T> __parameter0);
```

### Constraint

T is MultiplicativeMonoid

### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.Multiplies<T> | |

### Returns

T

Returns $T(1)$.

### 2.4.99 IdentityElement(System.Plus<T>) Function

Returns the identity element of addition, that is: T(0).

#### Syntax

```
public T IdentityElement(System.Plus<T> __parameter0);
```

#### Constraint

T is AdditiveMonoid

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.Plus<T> | |

#### Returns

T

Returns $T(0)$.

## 2.4.100 Inserter(C&, I) Function

Returns an InsertIterator<C> for an insertion sequence and its iterator.

### Syntax

```
public System.InsertIterator<C> Inserter(C& c, I i);
```

### Constraint

C is InsertionSequence and I is C.Iterator

### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | An insertion sequence. |
| i | I | An iterator pointing a to position to insert elements. |

### Returns

System.InsertIterator<C>

Returns an InsertIterator<C> for an insertion sequence and iterator.

### Remarks

An InsertIterator<C> is an output iterator that inserts elements to some position of an insertion sequence.

### 2.4.100.1 Example

```
using System;
using System.Collections;

//  Writes:
//  0, 1, 2, 3, 4

void main()
{
    Set<int> s;
    s.Insert(3);
    s.Insert(2);
    s.Insert(1);
//  Set is a sorted container, so it contains now 1, 2, 3...

    List<int> list;
    list.Add(0);
    list.Add(4);

//  Copy the set in the middle of the list using InsertIterator...
    Copy(s.CBegin(), s.CEnd(), Inserter(list, list.Begin() + 1));

//  ForwardContainers containing ints can be put to OutputStream...
    Console.Out() << list << endl();
}
```

## 2.4.101 InsertionSort(I, I) Function

Sorts a sequence of values using insertion sort algorithm.

### Syntax

```
public void InsertionSort(I begin, I end);
```

### Constraint

I is RandomAccessIterator and I.ValueType is TotallyOrdered

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A random access iterator pointing to the beginning of a sequence. |
| end | I | A random access iterator pointing one past the end of a sequence. |

#### 2.4.101.1 Example

```
using System;
using System.Collections;

//  Writes:
//  -1, 0, 4, 4, 6, 7

void main()
{
    List<int> list;
    list.Add(7);
    list.Add(-1);
    list.Add(4);
    list.Add(6);
    list.Add(0);
    list.Add(4);
    InsertionSort(list.Begin(), list.End());
    Console.Out() << list << endl();
}
```

### Implementation

algorithm.cm, page 11

## 2.4.102 InsertionSort(I, I, R) Function

Sorts a sequence of values using insertion sort algorithm and given ordering relation.

### Syntax

```
public void InsertionSort(I begin, I end, R r);
```

### Constraint

I is RandomAccessIterator and R is Relation and R.Domain is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A random access iterator pointing to the beginning of a sequence. |
| end | I | A random access iterator pointing one past the end of a sequence. |
| r | R | An ordering relation. |

### 2.4.102.1 Example

```
using System;
using System.Collections;

//  Writes:
//  7, 6, 4, 4, 0, -1

void main()
{
    List<int> list;
    list.Add(7);
    list.Add(-1);
    list.Add(4);
    list.Add(6);
    list.Add(0);
    list.Add(4);
    InsertionSort(list.Begin(), list.End(), Greater<int>());
    Console.Out() << list << endl();
}
```

### Implementation

algorithm.cm, page 11

### 2.4.103 IsAlpha(char) Function

Returns true if the given character is an alphabetic character, false otherwise.

#### Syntax

```
public bool IsAlpha(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is an alphabetic character, false otherwise.

#### Remarks

Alphabetic characters are the lower case and upper case letters 'a' .. 'z' and 'A' ..'Z'.

### 2.4.104 IsAlphanumeric(char) Function

Returns true if the given character is an alphanumeric character, false otherwise.

#### Syntax

```
public bool IsAlphanumeric(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is an alphanumeric character, false otherwise.

#### Remarks

Alphanumeric characters are the lower case and upper case letters and decimal digits 'a'..'z', 'A'..'Z' and '0'..'9'.

### 2.4.105 IsControl(char) Function

Returns true if the given character is a control character, false otherwise.

#### Syntax

```
public bool IsControl(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is a control character, false otherwise.

#### Remarks

The control characters are characters whose ASCII codes are 0 .. 31 and 127.

### 2.4.106 IsDigit(char) Function

Returns true if the given character is a decimal digit, false otherwise.

#### Syntax

```
public bool IsDigit(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is a decimal digit, false otherwise.

#### Remarks

Decimal digits are '0' .. '9'.

## 2.4.107 IsGraphic(char) Function

Returns true if the given character is a graphical character, false otherwise.

### Syntax

```
public bool IsGraphic(char c);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

### Returns

bool

Returns true if the given character is a graphical character, false otherwise.

### Remarks

The graphical characters are characters whose ASCII codes are in the range 33 .. 126.

### 2.4.108 IsHexDigit(char) Function

Returns true if the given character is a hexadecimal digit, false otherwise.

#### Syntax

```
public bool IsHexDigit(char c);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

## Returns

bool

Returns true if the given character is a hexadecimal digit, false otherwise.

## Remarks

Hexadecimal digits are '0' .. '9', 'a' .. 'f' and 'A' ..'F'.

### 2.4.109 IsLower(char) Function

Returns true if the given character is a lower case letter, false otherwise.

#### Syntax

```
public bool IsLower(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is a lower case letter, false otherwise.

#### Remarks

Lower case letters are 'a' .. 'z'.

### 2.4.110 IsPrintable(char) Function

Returns true if the given character is a printable character, false otherwise.

#### Syntax

```
public bool IsPrintable(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is a printable character, false otherwise.

#### Remarks

Printable characters are characters whose ASCII codes are in the range 32 .. 126.

### 2.4.111 IsPunctuation(char) Function

Returns true if the given character is a punctuation character, false otherwise.

#### Syntax

```
public bool IsPunctuation(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is a punctuation character, false otherwise.

#### Remarks

Punctuation characters are characters whose ASCII codes are in ranges 33 .. 47, 58 .. 64, 91 .. 96 and 123 .. 126.

### 2.4.112 IsSpace(char) Function

Returns true if the given character is a space character, false otherwise.

#### Syntax

```
public bool IsSpace(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is a space character, false otherwise.

#### Remarks

The space characters are characters whose ASCII codes are 9 .. 13 and 32.

### 2.4.113 IsUpper(char) Function

Returns true if the given character is an upper case letter, false otherwise.

#### Syntax

```
public bool IsUpper(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to test. |

#### Returns

bool

Returns true if the given character is an upper case letter, false otherwise.

#### Remarks

Upper case letters are 'A' .. 'Z'.

### 2.4.114 LastComponentsEqual(const System.String<CharT>&, const System.String<CharT>&, CharT) Function

Returns true, if last components of two strings are equal, false otherwise.

#### Syntax

```
public bool LastComponentsEqual(const System.String<CharT>& s0, const System.String<CharT>&
s1, CharT componentSeparator);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s0 | const System.String<CharT>& | First string. |
| s1 | const System.String<CharT>& | Second string. |
| componentSeparator | CharT | Component separator character. |

#### Returns

bool

Returns true, if last components of two strings are equal, false otherwise.

#### Remarks

First splits s0 and s1 to components with respect to component separator character componentSeparator. Then computes $n$, the minumum number of components of s0 and s1. Then returns true, if last $n$ components of s0 and s1 are equal, false otherwise.

## 2.4.115 LexicographicalCompare(I1, I1, I2, I2) Function

Returns true if the first sequence comes lexicographically before the second sequence, false otherwise.

### Syntax

```
public bool LexicographicalCompare(I1 first1, I1 last1, I2 first2, I2 last2);
```

### Constraint

I1 is InputIterator and I2 is InputIterator and LessThanComparable<I1.ValueType, I2.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| first1 | I1 | An input iterator pointing to the beginning of the first sequence. |
| last1 | I1 | An input iterator pointing one past the end of the first sequence. |
| first2 | I2 | An input iterator pointing to the beginning of the second sequence. |
| last2 | I2 | An input iterator pointing one past the end of the second sequence. |

### Returns

bool

Returns true if the first sequence comes lexicographically before the second sequence, false otherwise.

### Remarks

Compares the sequences element by element, until (1) an element in the first sequence is less than the corresponding element in the second sequence. In that case returns true. (2) an element in the second sequence is less than the corresponding element in the first sequence. In that case returns false. (3) The corresponding elements of the sequences are equal, but the first sequence is shorter than the second sequence. In that case returns true. (4) Otherwise returns false.

### 2.4.115.1 Example

```
using System;
using System.Collections;

//  Writes:
//  I thought so

void main()
{
```

```
    List<string> fruits1;
    fruits1.Add("apple");
    fruits1.Add("banana");
    fruits1.Add("grape");

    List<string> fruits2;
    fruits2.Add("apple");
    fruits2.Add("banana");
    fruits2.Add("grape");
    fruits2.Add("orange");

//  fruits1 comes lexicographically before fruits2, so...

    if (LexicographicalCompare(fruits1.CBegin(), fruits1.CEnd(), fruits2.
        CBegin(), fruits2.CEnd()))
    {
        Console.Out() << "I thought so" << endl();
    }
    if (LexicographicalCompare(fruits2.CBegin(), fruits2.CEnd(), fruits1.
        CBegin(), fruits1.CEnd()))
    {
        Console.Error() << "bug" << endl();
    }
}
```

## Implementation

algorithm.cm, page 13

## 2.4.116 LexicographicalCompare(I1, I1, I2, I2, R) Function

Returns true if the first sequence comes lexicographically before the second sequence according to the given ordering relation, false otherwise.

### Syntax

```
public bool LexicographicalCompare(I1 first1, I1 last1, I2 first2, I2 last2, R r);
```

### Constraint

I1 is InputIterator and I2 is InputIterator and Same<I1.ValueType, I2.ValueType> and Relation<R, I1.ValueType, I2.ValueType> and Relation<R, I2.ValueType, I1.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| first1 | I1 | An input iterator pointing to the beginning of the first sequence. |
| last1 | I1 | An input iterator pointing one past the end of the first sequence. |
| first2 | I2 | An input iterator pointing to the beginning of the second sequence. |
| last2 | I2 | An input iterator pointing one past the end of the second sequence. |
| r | R | An ordering relation. |

### Returns

bool

Returns true if the first sequence comes lexicographically before the second sequence according to the given ordering relation, false otherwise.

### Remarks

Compares the sequences element by element, until (1) an element in the first sequence is less than the corresponding element in the second sequence according to the given ordering relation. In that case returns true. (2) an element in the second sequence is less than the corresponding element in the first sequence according to the given ordering relation. In that case returns false. (3) The corresponding elements of the sequences are equal, but the first sequence is shorter than the second sequence. In that case returns true. (4) Otherwise returns false.

### 2.4.116.1 Example

```
using System;
using System.Collections;

//  Writes:
```

```
//  list comes lexicographically before set

public class A
{
    public A(): id(0)
    {
    }
    public A(int id_): id(id_)
    {
    }
    public nothrow inline int Id() const
    {
        return id;
    }
    private int id;
}

public class ALess: Rel<A>
{
    public nothrow inline bool operator()(const A& left, const A& right)
        const
    {
        return left.Id() < right.Id();
    }
}

void main()
{
    List<A> list;
    list.Add(A(3));
    list.Add(A(4));
    list.Add(A(5));

    Set<A, ALess> set;
    set.Insert(A(3));
    set.Insert(A(4));
    set.Insert(A(6));

    if (LexicographicalCompare(list.CBegin(), list.CEnd(), set.CBegin(), set
        .CEnd(), ALess()))
    {
        Console.Out() << "list comes lexicographically before set" << endl()
            ;
    }
    else
    {
        Console.Out() << "bug" << endl();
    }
}
```

## Implementation

algorithm.cm, page 13

## 2.4.117 LowerBound(I, I, const T&) Function

Finds a position of the first element in a sorted sequence that is greater than or equal to the given value.

### Syntax

```
public I LowerBound(I first, I last, const T& value);
```

### Constraint

I is ForwardIterator and TotallyOrdered<T, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator pointing to the beginning of a sorted sequence. |
| last | I | A forward iterator pointing one past the end of a sorted sequence. |
| value | const T& | A value to search. |

### Returns

I

Returns an iterator pointing to the first element in a sorted sequence that is greater than or equal to the given value, if there is one, otherwise returns *last*.

### Remarks

Uses a binary search algorithm to search the position.

#### 2.4.117.1 Example

```
using System;
using System.Collections;

//  Writes:
//  position of "Stroustrup" is 3

void main()
{
    List<string> persons;
    persons.Add("Stroustrup, Bjarne");
    persons.Add("Stepanov, Alexander");
    persons.Add("Knuth, Donald E.");
    persons.Add("Dijkstra, Edsger W.");
    persons.Add("Turing, Alan");
    Sort(persons);
    List<string>.ConstIterator s = LowerBound(persons.CBegin(), persons.CEnd
        (), string("Stroustrup"));
```

```
    if (s != persons.CEnd())
    {
        Console.Out() << "position of \"Stroustrup\" is " << Distance(
            persons.CBegin(), s) << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

## Implementation

algorithm.cm, page 4

## 2.4.118 LowerBound(I, I, const T&, R) Function

Finds a position of the first element in a sorted sequence that is greater than or equal to the given value according to the given ordering relation.

### Syntax

```
public I LowerBound(I first, I last, const T& value, R r);
```

### Constraint

I is ForwardIterator and T is I.ValueType and R is Relation and R.Domain is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator pointing to the beginning of a sorted sequence. |
| last | I | A forward iterator pointing one past the end of a sorted sequence. |
| value | const T& | A value to search. |
| r | R | An ordering relation. |

### Returns

I

Returns an iterator pointing to the first element in a sorted sequence that is greater than or equal to the given value according to the given ordering relation, if there is one, otherwise returns *last*.

### Remarks

Uses a binary search algorithm to search the position.

### Implementation

algorithm.cm, page 4

### 2.4.119 MakePair(const T&, const U&) Function

Returns a pair composed of the given values.

#### Syntax

```
public System.Pair<T, U> MakePair(const T& first, const U& second);
```

#### Constraint

T is Semiregular and U is Semiregular

#### Parameters

| Name | Type | Description |
|---|---|---|
| first | const T& | The first value. |
| second | const U& | The second value. |

#### Returns

System.Pair<T, U>

Returns a pair composed of the given values.

### 2.4.120 Max(const T&, const T&) Function

Returns the maximum of two values.

#### Syntax

```
public const T& Max(const T& left, const T& right);
```

#### Constraint

T is LessThanComparable

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first value. |
| right | const T& | The second value. |

#### Returns

const T&

If $right \geq left$ returns $right$, else returns $left$.

## 2.4.121 MaxElement(I, I) Function

Returns the position of the first occurrence of the largest element in a sequence of elements.

### Syntax

```
public I MaxElement(I first, I last);
```

### Constraint

I is ForwardIterator and I.ValueType is TotallyOrdered

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator pointing to the beginning of sequence. |
| last | I | A forward iterator pointing one past the end of a sequence. |

### Returns

I

Returns the position of the first occurrence of the largest element in a sequence of elements.

#### 2.4.121.1 Example

```
using System;
using System.Collections;

//  Writes:
//  maximum element is 2 at position 3

void main()
{
    List<int> list;
    list.Add(1);
    list.Add(0);
    list.Add(1);
    list.Add(2);
    list.Add(0);
    list.Add(2);
    List<int>.ConstIterator maxPos = MaxElement(list.CBegin(), list.CEnd());
    if (maxPos != list.CEnd())
    {
        Console.Out() << "maximum element is " << *maxPos << " at position "
            << Distance(list.CBegin(), maxPos) << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

### Implementation

algorithm.cm, page 14

## 2.4.122 MaxElement(I, I, R) Function

Returns the position of the first occurrence of the largest element according to the given ordering relation in a sequence of elements.

### Syntax

```
public I MaxElement(I first, I last, R r);
```

### Constraint

I is ForwardIterator and R is Relation and R.Domain is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator pointing to the beginning of sequence. |
| last | I | A forward iterator pointing one past the end of a sequence. |
| r | R | An ordering relation. |

### Returns

I

Returns the position of the first occurrence of the largest element according to the given ordering relation in a sequence of elements.

### Implementation

algorithm.cm, page 15

### 2.4.123 MaxValue() Function

Returns the largest value of an integer type.

**Syntax**

```
public I MaxValue();
```

## Returns

I

Returns the largest value of an integer type.

### 2.4.124 MaxValue(byte) Function

Returns the maximum value of **byte**: 255.

#### Syntax

```
public byte MaxValue(byte __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | byte | |

#### Returns

byte

Returns the maximum value of **byte**: 255.

### 2.4.125 MaxValue(int) Function

Returns the maximum value of **int**: 2147483647.

#### Syntax

```
public int MaxValue(int __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | int | |

#### Returns

int

Returns the maximum value of **int**: 2147483647.

### 2.4.126 MaxValue(long) Function

Returns the maximum value of **long**: 9223372036854775807.

#### Syntax

```
public long MaxValue(long __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | long | |

#### Returns

long

Returns the maximum value of **long**: 9223372036854775807.

### 2.4.127 MaxValue(sbyte) Function

Returns the maximum value of **sbyte**: 127.

#### Syntax

```
public sbyte MaxValue(sbyte __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | sbyte | |

#### Returns

sbyte

Returns the maximum value of **sbyte**: 127.

### 2.4.128 MaxValue(short) Function

Returns the maximum value of **short**: 32767.

#### Syntax

```
public short MaxValue(short __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | short | |

#### Returns

short

Returns the maximum value of **short**: 32767.

### 2.4.129 MaxValue(uint) Function

Returns the maximum value of **uint**: 4294967295.

#### Syntax

```
public uint MaxValue(uint __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | uint | |

#### Returns

uint

Returns the maximum value of **uint**: 4294967295.

### 2.4.130 MaxValue(ulong) Function

Returns the maximum value of **ulong**: 18446744073709551615.

#### Syntax

```
public ulong MaxValue(ulong __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | ulong | |

#### Returns

ulong

Returns the maximum value of **ulong**: 18446744073709551615.

### 2.4.131 MaxValue(ushort) Function

Returns the maximum value of **ushort**: 65535.

#### Syntax

```
public ushort MaxValue(ushort __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | ushort | |

#### Returns

ushort

Returns the maximum value of **ushort**: 65535.

### 2.4.132 Median(const T&, const T&, const T&) Function

Returns the median of three values.

#### Syntax

```
public const T& Median(const T& a, const T& b, const T& c);
```

#### Constraint

T is TotallyOrdered

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first value. |
| b | const T& | The second value. |
| c | const T& | The third value. |

#### Returns

const T&

Returns the median of three values.

### 2.4.133 Median(const T&, const T&, const T&, R) Function

Returns the median of three values according to the given ordering relation.

#### Syntax

```
public const T& Median(const T& a, const T& b, const T& c, R r);
```

#### Constraint

T is Semiregular and R is Relation and R.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first value. |
| b | const T& | The second value. |
| c | const T& | The third value. |
| r | R | An ordering relation. |

#### Returns

const T&

Returns the median of three values according to the given ordering relation.

### 2.4.134 Min(const T&, const T&) Function

Returns the minimum of two values.

#### Syntax

```
public const T& Min(const T& left, const T& right);
```

#### Constraint

T is LessThanComparable

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const T& | The first value. |
| right | const T& | The second value. |

#### Returns

const T&

If $left \leq right$ returns $left$, else returns $right$.

## 2.4.135 MinElement(I, I) Function

Returns the position of the first occurrence of the smallest element in a sequence of elements.

### Syntax

```
public I MinElement(I first, I last);
```

### Constraint

I is ForwardIterator and I.ValueType is TotallyOrdered

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator pointing to the beginning of a sequence. |
| last | I | A forward iterator pointing one past the end of a sequence. |

### Returns

I

Returns the position of the first occurrence of the smallest element in a sequence of elements.

#### 2.4.135.1 Example

```
using System;
using System.Collections;

//  Writes:
//  minimum element is 0 at position 1

void main()
{
    List<int> list;
    list.Add(1);
    list.Add(0);
    list.Add(1);
    list.Add(2);
    list.Add(0);
    list.Add(2);
    List<int>.ConstIterator minPos = MinElement(list.CBegin(), list.CEnd());
    if (minPos != list.CEnd())
    {
        Console.Out() << "minimum element is " << *minPos << " at position "
            << Distance(list.CBegin(), minPos) << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

### Implementation

algorithm.cm, page 13

## 2.4.136 MinElement(I, I, R) Function

Returns the position of the first occurrence of the smallest element according to the given ordering relation in a sequence of elements.

### Syntax

```
public I MinElement(I first, I last, R r);
```

### Constraint

I is ForwardIterator and R is Relation and R.Domain is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator pointing to the beginning of a sequence. |
| last | I | A forward iteraor pointing one past the end of a sequence. |
| r | R | An ordering relation. |

### Returns

I

Returns the position of the first occurrence of the smallest element according to the given ordering relation in a sequence of elements.

### Implementation

algorithm.cm, page 14

### 2.4.137 MinValue() Function

Returns the smallest value of an integer type.

#### Syntax

```
public I MinValue();
```

#### Returns

I

Returns the smallest value of an integer type.

### 2.4.138 MinValue(byte) Function

Returns the minimum value of **byte**: 0.

#### Syntax

```
public byte MinValue(byte __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | byte | |

#### Returns

byte

Returns the minimum value of **byte**: 0.

### 2.4.139 MinValue(int) Function

Returns the minimum value of **int**: -2147483648.

#### Syntax

```
public int MinValue(int __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | int | |

#### Returns

int

Returns the minimum value of **int**: -2147483648.

### 2.4.140 MinValue(long) Function

Returns the minimum value of **long**: -9223372036854775808.

#### Syntax

```
public long MinValue(long __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | long | |

#### Returns

long

Returns the minimum value of **long**: -9223372036854775808.

### 2.4.141 MinValue(sbyte) Function

Returns the minimum value of **sbyte**: -128.

#### Syntax

```
public sbyte MinValue(sbyte __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | sbyte | |

#### Returns

sbyte

Returns the minimum value of **sbyte**: -128.

### 2.4.142 MinValue(short) Function

Returns the minimum value of **short**: -32768.

#### Syntax

```
public short MinValue(short __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | short | |

#### Returns

short

Returns the minimum value of **short**: -32768.

### 2.4.143 MinValue(uint) Function

Returns the minimum value of **uint**: 0.

#### Syntax

```
public uint MinValue(uint __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | uint | |

#### Returns

uint

Returns the minimum value of **uint**: 0.

### 2.4.144 MinValue(ulong) Function

Returns the minimum value of **ulong**: 0.

#### Syntax

```
public ulong MinValue(ulong __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | ulong | |

#### Returns

ulong

Returns the minimum value of **ulong**: 0.

### 2.4.145 MinValue(ushort) Function

Returns the minimum value of **ushort**: 0.

#### Syntax

```
public ushort MinValue(ushort __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | ushort | |

#### Returns

ushort

Returns the minimum value of **ushort**: 0.

## 2.4.146 Move(I, I, O) Function

Moves a sequence.

### Syntax

```
public O Move(I begin, I end, O to);
```

### Constraint

I is InputIterator and O is OutputIterator and O.ValueType is I.ValueType and I.ValueType is MoveAssignable

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of a source sequence. |
| end | I | An input iterator pointing one past the end of a source sequence. |
| to | O | An output iterator pointing to the beginning of the target sequence. |

### Returns

O

Returns an output iterator pointing one past the end of the target sequence.

### Implementation

algorithm.cm, page 2

## 2.4.147 MoveBackward(I, I, O) Function

Moves a source sequence to a target sequence starting from the end of the source sequence.

### Syntax

```
public O MoveBackward(I begin, I end, O to);
```

### Constraint

I is BidirectionalIterator and O is BidirectionalIterator and O.ValueType is I.ValueType and I.ValueType is MoveAssignable

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A bidirectional iterator pointing to the beginning of the source sequence. |
| end | I | A bidirectional iterator pointing one pasth the end of the source sequence. |
| to | O | A bidirectional iterator pointing to the beginning of the target sequence. |

### Returns

O

Returns a bidirectional iterator pointing to the beginning of the target sequence.

### Implementation

algorithm.cm, page 3

### 2.4.148 Next(I, int) Function

Returns a random access iterator advanced the specified offset.

#### Syntax

```
public I Next(I i, int n);
```

#### Constraint

I is RandomAccessIterator

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | I | A random access iterator. |
| n | int | An offset to advance. |

#### Returns

I

Returns $i + n$.

#### Implementation

algorithm.cm, page 4

### 2.4.149 Next(I, int) Function

Returns a forward iterator advanced the specified number of steps.

#### Syntax

```
public I Next(I i, int n);
```

#### Constraint

I is ForwardIterator

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | I | A forward iterator. |
| n | int | A non-negative number of steps to advance. |

#### Returns

I

Returns a forward iterator advanced the specified number of steps.

#### Implementation

algorithm.cm, page 3

## 2.4.150 NextPermutation(I, I) Function

Computes the lexicographically next permutation of a sequence of elements.

### Syntax

```
public bool NextPermutation(I begin, I end);
```

### Constraint

I is BidirectionalIterator and I.ValueType is LessThanComparable

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A bidirectional iterator pointing to the beginning of a sequence. |
| end | I | A bidirectional iterator pointing one past the end of a sequence. |

### Returns

bool

Returns true if the permutation was not last permutation, false otherwise. If the permutation was last, the permutation returned is the lexicographically first permutation of the sequence.

### 2.4.150.1 Example

```
using System;
using System.Collections;

//  Writes:
//  1, 2, 3
//  1, 3, 2
//  2, 1, 3
//  2, 3, 1
//  3, 1, 2
//  3, 2, 1

void main()
{
    List<int> list;
    list.Add(1);
    list.Add(2);
    list.Add(3);
    Console.Out() << list << endl();
    while (NextPermutation(list.Begin(), list.End()))
    {
        Console.Out() << list << endl();
    }
}
```

### Implementation

algorithm.cm, page 16

### 2.4.151 NextPermutation(I, I, R) Function

Computes the lexicographically next permutation of a sequence of elements according to the given ordering relation.

#### Syntax

```
public bool NextPermutation(I begin, I end, R r);
```

#### Constraint

I is BidirectionalIterator and R is Relation and R.Domain is I.ValueType

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A bidirectional iterator pointing to the beginning of a sequence. |
| end | I | A bidirectional iterator pointing one past the end of a sequence. |
| r | R | An ordering relation. |

#### Returns

bool

Returns true if the permutation was not last permutation, false otherwise. If the permutation was last, the permutation returned is the lexicographically first permutation of the sequence.

#### Implementation

algorithm.cm, page 16

### 2.4.152 Now() Function

Returns current time point value from computer's real time clock.

#### Syntax

```
public System.TimePoint Now();
```

#### Returns

System.TimePoint

Returns number of nanoseconds elapsed since 1.1.1970 as a time point value.

### 2.4.153 ParseBool(const System.String<char>&) Function

Parses a Boolean value "true" or "false" from the given string and returns it.

#### Syntax

```
public bool ParseBool(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

#### Returns

bool

Returns true if *s* contains "true", false if *s* contains "false". Otherwise throws ConversionException.

### 2.4.154 ParseBool(const System.String<char>&, bool&) Function

Parses a Boolean value "true" or "false" from the given string and returns true if the parsing was successful, false if not.

#### Syntax

```
public bool ParseBool(const System.String<char>& s, bool& b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| b | bool& | If *s* contains "true" b is set to true, if *s* contains "false" b is set to false. |

#### Returns

bool

Returns true if the parsing was successful, false if not.

### 2.4.155 ParseDate(const System.String<char>&) Function

Parses a date from the given string and returns it.

#### Syntax

```
public System.Date ParseDate(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

#### Returns

System.Date

Returns the parsed date value if the parsing was successful. Otherwise throws ConversionException.

### 2.4.156 ParseDouble(const System.String<char>&) Function

Parses a **double** value from the given string and returns it.

#### Syntax

```
public double ParseDouble(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

#### Returns

double

Returns the parsed **double** value if the parsing was successful. Otherwise throws ConversionException.

### 2.4.157 ParseDouble(const System.String<char>&, double&) Function

Parses a **double** value from the given string and returns true if the parsing was successful, false if not.

#### Syntax

```
public bool ParseDouble(const System.String<char>& s, double& x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| x | double& | A **double** parsed from *s*. |

#### Returns

bool

Returns true if the parsing was successful, false if not.

## 2.4.158 ParseHex(const System.String<char>&) Function

Parses a hexadecimal value from a string.

### Syntax

```
public ulong ParseHex(const System.String<char>& s);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

### Returns

ulong

Returns the parsed hexadecimal value if the parsing was successful, otherwise throws ConversionException.

### 2.4.159 ParseHex(const System.String<char>&, System.uhuge&) Function

Parses a hexadecimal 128-bit value from a string and returns true, if the parsing was successful.

#### Syntax

```
public bool ParseHex(const System.String<char>& s, System.uhuge& hex);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| hex | System.uhuge& | Parsed 128-bit value. |

#### Returns

bool

Returns true, if the parsing was successful, false otherwise.

### 2.4.160 ParseHex(const System.String<char>&, ulong&) Function

Parses a hexadecimal value from a string and returns true if the parsing was successful.

#### Syntax

```
public bool ParseHex(const System.String<char>& s, ulong& hex);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| hex | ulong& | Parsed value. |

#### Returns

bool

Returns true, if the parsing was successful, false otherwise.

### 2.4.161 ParseHexUHuge(const System.String<char>&) Function

Parses a 128-bit decimal value from a string.

#### Syntax

```
public System.uhuge ParseHexUHuge(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

#### Returns

System.uhuge

Returns the parsed 128-bit decimal value if the parsing was successful, otherwise throws ConversionException.

### 2.4.162 ParseInt(const System.String<char>&) Function

Parses an **int** from the given string and returns it.

#### Syntax

```
public int ParseInt(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

#### Returns

int

Returns the parsed **int** value if the parsing was successful. Otherwise throws ConversionException.

### 2.4.163 ParseInt(const System.String<char>&, int&) Function

Parses an **int** value from the given string and returns true if the parsing was successful, false if not.

#### Syntax

```
public bool ParseInt(const System.String<char>& s, int& x);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| x | int& | An **int** parsed from *s*. |

## Returns

bool

Returns true if the parsing was successful, false if not.

### 2.4.164 ParseLong(const System.String<char>&) Function

Parses a **long** from the given string and returns it.

#### Syntax

```
public long ParseLong(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

#### Returns

long

Returns the parsed **long** value if the parsing was successful. Otherwise throws ConversionException.

### 2.4.165 ParseLong(const System.String<char>&, long&) Function

Parses a **long** value from the given string and returns true if the parsing was successful, false if not.

#### Syntax

```
public bool ParseLong(const System.String<char>& s, long& x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| x | long& | A **long** parsed from *s*. |

#### Returns

bool

Returns true if the parsing was successful, false if not.

### 2.4.166 ParseUHuge(const System.String<char>&) Function

Parses a decimal 128-bit value from a string.

#### Syntax

```
public System.uhuge ParseUHuge(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

#### Returns

System.uhuge

Returns parsed 128-bit decimal value if parsing was successful, otherwise throws ConversionException.

### 2.4.167 ParseUHuge(const System.String<char>&, System.uhuge&) Function

Parses a decimal 128-bit value from a string and returns true, if the parsing was successful.

#### Syntax

```
public bool ParseUHuge(const System.String<char>& s, System.uhuge& x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| x | System.uhuge& | Parsed 128-bit value. |

#### Returns

bool

Returns true, if the parsing was successful, false otherwise.

### 2.4.168 ParseUInt(const System.String<char>&) Function

Parses an **uint** from the given string and returns it.

#### Syntax

```
public uint ParseUInt(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

#### Returns

uint

Returns the parsed **uint** value if the parsing was successful. Otherwise throws ConversionException.

### 2.4.169 ParseUInt(const System.String<char>&, uint&) Function

Parses an **uint** value from the given string and returns true if the parsing was successful, false if not.

#### Syntax

```
public bool ParseUInt(const System.String<char>& s, uint& x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| x | uint& | An **uint** parsed from *s*. |

#### Returns

bool

Returns true if the parsing was successful, false if not.

### 2.4.170 ParseULong(const System.String<char>&) Function

Parses an **ulong** from the given string and returns it.

#### Syntax

```
public ulong ParseULong(const System.String<char>& s);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |

## Returns

ulong

Returns the parsed **ulong** value if the parsing was successful. Otherwise throws ConversionException.

### 2.4.171 ParseULong(const System.String<char>&, ulong&) Function

Parses an **ulong** value from the given string and returns true if the parsing was successful, false if not.

#### Syntax

```
public bool ParseULong(const System.String<char>& s, ulong& x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to parse. |
| x | ulong& | An **ulong** parsed from *s*. |

#### Returns

bool

Returns true if the parsing was successful, false if not.

## 2.4.172 PrevPermutation(I, I) Function

Computes the lexicographically previous permutation of a sequence of elements.

### Syntax

```
public bool PrevPermutation(I begin, I end);
```

### Constraint

I is BidirectionalIterator and I.ValueType is LessThanComparable

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A bidirectional iterator pointing to the beginning of a sequence. |
| end | I | A bidirectional iterator pointing one past the end of a sequence. |

### Returns

bool

Returns true if the permutation was not first permutation, false otherwise. If the permutation was first, the permutation returned is the lexicographically last permutation of the sequence.

### 2.4.173 PrevPermutation(I, I, R) Function

Computes the lexicographically previous permutation according to the given ordering relation of a sequence of elements.

#### Syntax

```
public bool PrevPermutation(I begin, I end, R r);
```

#### Constraint

I is BidirectionalIterator and R is Relation and R.Domain is I.ValueType

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A bidirectional iterator pointing to the beginning of a sequence. |
| end | I | A bidirectional iterator pointing one past the end of a sequence. |
| r | R | An ordering relation. |

#### Returns

bool

Returns true if the permutation was not first permutation, false otherwise. If the permutation was first, the permutation returned is the lexicographically last permutation of the sequence.

## 2.4.174 PtrCast(const System.SharedPtr<T>&) Function

Casts a shared pointer.

### Syntax

```
public System.SharedPtr<U> PtrCast(const System.SharedPtr<T>& from);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| from | const System.SharedPtr<T>& | A shared pointer to cast from. |

### Returns

System.SharedPtr<U>

Returns the shared pointer casted to $SharedPtr < U >$.

#### 2.4.174.1 Example

```
using System;

public class Base
{
    public virtual ~Base()
    {
    }
}

public class Derived: Base
{
}

void main()
{
    SharedPtr<Base> ptr(new Derived());
    SharedPtr<Derived> derived = PtrCast<Derived>(ptr);
}
```

### 2.4.175 Rand() Function

Returns a pseudorandom number generated by the Mersenne Twister pseudorandom number generator MT.

#### Syntax

```
public uint Rand();
```

#### Returns

uint

Returns a pseudorandom number generated by the Mersenne Twister pseudorandom number generator MT.

#### Remarks

The returned pseudorandom number is in range 0..MaxValue(uint) inclusive.

### 2.4.176 RandomNumber(uint) Function

Returns a pseudorandom number in range 0..n - 1 inclusive.

#### Syntax

```
public uint RandomNumber(uint n);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| n | uint | Upper bound for the generated pseudorandom number. |

#### Returns

uint

Returns a pseudorandom number in range 0..n - 1 inclusive.

#### Remarks

Uses Rand() function to generate a pseudorandom number and returns that number modulo n.

### 2.4.177 RandomShuffle(I, I) Function

Computes a random permutation of a random access sequence.

#### Syntax

```
public void RandomShuffle(I begin, I end);
```

#### Constraint

I is RandomAccessIterator

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An iterator pointing to the beginning of the sequence. |
| end | I | An iterator pointing one past the end of the sequence. |

## 2.4.178 Remove(I, I, P) Function

Removes values satisfying a given predicate from a sequence by moving them to the end of the sequence. Returns an iterator pointing to the beginning of removed values.

### Syntax

```
public I Remove(I begin, I end, P p);
```

### Constraint

I is ForwardIterator and P is UnaryPredicate and P.ArgumentType is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An iterator pointing to the beginning of a sequence. |
| end | I | An iterator pointing to one past the end of a sequence. |
| p | P | A unary predicate. |

### Returns

I

Returns an iterator pointing to the beginning of removed values.

## 2.4.179 Remove(I, I, const T&) Function

Removes the values equal to the given value from the given sequence of values by moving them to the end of the sequence. Returns an iterator pointing to the beginning of removed values.

### Syntax

```
public I Remove(I begin, I end, const T& value);
```

### Constraint

I is ForwardIterator and T is Semiregular and EqualityComparable<T, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An iterator pointing to the beginning of a sequence. |
| end | I | An iterator pointing one past the end of a sequence. |
| value | const T& | A value to remove. |

### Returns

I

Returns an iterator pointing to the beginning of removed values.

## 2.4.180 RemoveCopy(I, I, O, P) Function

Copies elements from given range to a destination range excluding elements satisfying a given predicate. Returns an iterator pointing one past the end of the destination range.

### Syntax

```
public O RemoveCopy(I begin, I end, O result, P p);
```

### Constraint

I is InputIterator and O is OutputIterator and O.ValueType is I.ValueType and P is UnaryPredicate and P.ArgumentType is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An iterator pointing to the beginning of a sequence to copy. |
| end | I | An iterator pointing one past the end of a sequence to copy. |
| result | O | An iterator pointing to the beginning of a destination range. |
| p | P | A unary predicate. |

### Returns

O

Returns an iterator pointing one past the end of the destination range.

## 2.4.181 RemoveCopy(I, I, O, const T&) Function

Copies elements from given range to a destination range excluding elements that are equal to given value. Returns an iterator pointing one past the end of the destination range.

### Syntax

```
public O RemoveCopy(I begin, I end, O result, const T& value);
```

### Constraint

T is Semiregular and I is InputIterator and O is OutputIterator and O.ValueType is I.ValueType and EqualityComparable<T, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An iterator pointing to the beginning of a sequence to copy. |
| end | I | An iterator pointing one past the end of a sequence to copy. |
| result | O | An iterator pointing to the beginning of a destination range. |
| value | const T& | A value. |

### Returns

O

Returns an iterator pointing one past the end of the destination range.

## 2.4.182 RethrowException(System.ExceptionPtr&) Function

Rethrows an exception captured to an ExceptionPtr.

### Syntax

```
public void RethrowException(System.ExceptionPtr& capturedException);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| capturedException | System.ExceptionPtr& | Exception captured to an ExceptionPtr. |

### Remarks

Used with function CaptureCurrentException().

#### 2.4.182.1 Example

```
using System;

// This program writes the following message to the standard error stream:
// FooException at 'C:/Temp/exptr/System.ExceptionPtr.cm' line 32:
// exception from thread
// call stack:
// 1> function 'foo()' file C:/Temp/exptr/System.ExceptionPtr.cm line 32
// 0> function 'ThreadFunction(void*)' file C:/Temp/exptr/System.
    ExceptionPtr.cm line 40

public class FooException : Exception
{
    public FooException(const string& message_) : base(message_)
    {
    }
}

public class ThreadData
{
    public void SetException(ExceptionPtr&& exPtr_)
    {
        exPtr = exPtr_;
    }
    public ExceptionPtr GetException() const
    {
        return Rvalue(exPtr);
    }
    private ExceptionPtr exPtr;
}

void foo()
{
    throw FooException("exception from thread");
}
```

```
void ThreadFunction(void* data)
{
    ThreadData* threadData = cast<ThreadData*>(data);
    try
    {
        foo();
    }
    catch (const Exception& ex)
    {
        ExceptionPtr exPtr = CaptureCurrentException();
        threadData->SetException(Rvalue(exPtr));
    }
}

void main()
{
    try
    {
        ThreadData threadData;
        System.Threading.Thread thread(System.Threading.ThreadFun(
            ThreadFunction), &threadData);
        thread.Join();
        ExceptionPtr exPtr = threadData.GetException();
        if (exPtr.HasException())
        {
            RethrowException(exPtr);
        }
    }
    catch (const Exception& ex)
    {
        Console.Error() << ex.ToString() << endl();
    }
}
```

## 2.4.183 Reverse(I, I) Function

Reverses a sequence.

### Syntax

```
public void Reverse(I begin, I end);
```

### Constraint

I is RandomAccessIterator

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A random access iterator pointing to the beginning of a sequence. |
| end | I | A random access iterator pointing one past the end of a sequence. |

### 2.4.183.1 Example

```
using System;
using System.Collections;

//  Writes:
//  3, 2, 1

void main()
{
    List<int> list;
    list.Add(1);
    list.Add(2);
    list.Add(3);
    Reverse(list.Begin(), list.End());
    Console.Out() << list << endl();
}
```

### Implementation

algorithm.cm, page 1

### 2.4.184 Reverse(I, I) Function

Reverses a sequence.

#### Syntax

```
public void Reverse(I begin, I end);
```

#### Constraint

I is BidirectionalIterator

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A bidirectional iterator pointing to the beginning of a sequence. |
| end | I | A bidirectional iterator pointing one past the end of a sequence. |

#### Implementation

algorithm.cm, page 2

## 2.4.185 ReverseUntil(I, I, I) Function

Reverses a sequence until the given middle iterator is hit.

### Syntax

```
public System.Pair<I, I> ReverseUntil(I first, I middle, I last);
```

### Constraint

I is BidirectionalIterator

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A bidirectional iterator pointing to the beginning of a sequence. |
| middle | I | A bidirectional iterator pointing a position between first and last inclusive. |
| last | I | A bidirectional iterator pointing one past the end of a sequence. |

### Returns

System.Pair<I, I>

Returns a pair of iterators. See remarks section.

### Remarks

While first and last differ from middle, decrements last, swaps elements pointed by first and last, and increments first. Then returns a pair made of first and last.

## 2.4.186 Rotate(I, I, I) Function

Rotates a sequence with respect to a middle iterator and returns an iterator pointing to the new middle.

### Syntax

```
public I Rotate(I first, I middle, I last);
```

### Constraint

I is BidirectionalIterator

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A bidirectional iterator pointing to the beginning of a sequence. |
| middle | I | An iterator pointing to a position between first and last incusive. |
| last | I | A bidirectional iterator pointing one past the end of a sequence. |

### Returns

I

Returns an iterator pointing to the new middle.

### Remarks

Shifts a sequence of elements pointed by iterators first and middle (where the middle iterator points one past the end of that range) to the end of the whole sequence first and last.

## 2.4.187 Rvalue(T&&) Function

Converts an argument to an rvalue so that it can be moved.

### Syntax

```
public T&& Rvalue(T&& x);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| x | T&& | A argument to convert. |

### Returns

T&&

Returns an rvalue reference of the argument.

### 2.4.187.1 Example

```
//  Writes:
//  10

using System;
using System.Collections;
using System.IO;

public class A
{
    public A(): ptr(null) {}
    public A(int x): ptr(new int(x)) {}
    public ~A()
    {
        delete ptr;
    }
    suppress A(const A&);
    suppress void operator=(const A&);
    public A(A&& that): ptr(that.ptr)
    {
        that.ptr = null;
    }
    public void operator=(A&& that)
    {
        Swap(ptr, that.ptr);
    }
    public int Value() const
    {
        #assert(ptr != null);
        return *ptr;
    }
    private int* ptr;
}

OutputStream& operator<<(OutputStream& s, const A& a)
{
    return s << a.Value() << endl();
```

```
}

void main()
{
    List<A> alist;
    A a(10);
    alist.Add(Rvalue(a));
    for (const A& a : alist)
    {
        Console.Out() << a << endl();
    }
}
```

### 2.4.188 Select_0_2(const T&, const T&, R) Function

Returns the smaller of two values according to the given ordering relation.

#### Syntax

```
public const T& Select_0_2(const T& a, const T& b, R r);
```

#### Constraint

T is Semiregular and R is Relation and R.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first value. |
| b | const T& | The second value. |
| r | R | An ordering relation. |

#### Returns

const T&

Returns the smaller of two values according to the given ordering relation.

### 2.4.189 Select_0_3(const T&, const T&, const T&, R) Function

Returns the smallest of tree values according to the given ordering relation.

#### Syntax

```
public const T& Select_0_3(const T& a, const T& b, const T& c, R r);
```

## Constraint

T is Semiregular and R is Relation and R.Domain is T

## Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first value. |
| b | const T& | The second value. |
| c | const T& | The third value. |
| r | R | An ordering relation. |

## Returns

const T&

Returns the smallest of tree values according to the given ordering relation.

### 2.4.190 Select_1_2(const T&, const T&, R) Function

Returns the larger of two values according to the given ordering relation.

#### Syntax

```
public const T& Select_1_2(const T& a, const T& b, R r);
```

#### Constraint

T is Semiregular and R is Relation and R.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first value. |
| b | const T& | The second value. |
| r | R | An ordering relation. |

#### Returns

const T&

Returns the larger of two values according to the given ordering relation.

### 2.4.191 Select_1_3(const T&, const T&, const T&, R) Function

Returns the median of three values according to the given ordering relation.

#### Syntax

```
public const T& Select_1_3(const T& a, const T& b, const T& c, R r);
```

#### Constraint

T is Semiregular and R is Relation and R.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first value. |
| b | const T& | The second value. |
| c | const T& | The third value. |
| r | R | An ordering relation. |

#### Returns

const T&

Returns the median of three values according to the given ordering relation.

### 2.4.192 Select_1_3_ab(const T&, const T&, const T&, R) Function

Returns the median of three values when the first two are in increasing order according to the given ordering relation.

#### Syntax

```
public const T& Select_1_3_ab(const T& a, const T& b, const T& c, R r);
```

#### Constraint

T is Semiregular and R is Relation and R.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first value. |
| b | const T& | The second value. |
| c | const T& | The third value. |
| r | R | An ordering relation. |

#### Returns

const T&

Returns the median of three values when the first two are in increasing order according to the given ordering relation.

### 2.4.193 Select_2_3(const T&, const T&, const T&, R) Function

Returns the largest of three values according to the given ordering relation.

#### Syntax

```
public const T& Select_2_3(const T& a, const T& b, const T& c, R r);
```

#### Constraint

T is Semiregular and R is Relation and R.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| a | const T& | The first value. |
| b | const T& | The second value. |
| c | const T& | The third value. |
| r | R | An ordering relation. |

#### Returns

const T&

Returns the largest of three values according to the given ordering relation.

## 2.4.194 Sort(C&) Function

Sorts the elements of a random access container to increasing order.

### Syntax

```
public void Sort(C& c);
```

### Constraint

C is RandomAccessContainer and C.Iterator.ValueType is TotallyOrdered

### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | A random access container. |

### 2.4.194.1 Example

```
using System;
using System.Collections;

//  Writes:
//  41, 6334, 11478, 15724, 18467, 19169, 24464, 26500, 26962, 29358

void main()
{
    List<int> list;
    for (int i = 0; i < 10; ++i)
    {
        list.Add(rand());
    }
    Sort(list);
    Console.Out() << list << endl();
}
```

### Implementation

algorithm.cm, page 12

## 2.4.195 Sort(C&) Function

Sorts the elements of a forward container to increasing order.

### Syntax

```
public void Sort(C& c);
```

### Constraint

C is ForwardContainer and C.Iterator.ValueType is TotallyOrdered

### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | A forward container, |

### Remarks

First the elements from the forward container are copied to a List<T>, then the List<T> is sorted, and finally the elements from the List<T> are copied back to the forward container.

### Implementation

algorithm.cm, page 12

## 2.4.196 Sort(C&, R) Function

Sorts the elements of a random access container to order according to the given ordering relation.

### Syntax

```
public void Sort(C& c, R r);
```

### Constraint

C is RandomAccessContainer and R is Relation and R.Domain is C.Iterator.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | A random access container. |
| r | R | An ordering relation. |

### 2.4.196.1 Example

```
using System;
using System.Collections;

//  Writes:
//  29358, 26962, 26500, 24464, 19169, 18467, 15724, 11478, 6334, 41

void main()
{
    List<int> list;
    for (int i = 0; i < 10; ++i)
    {
        list.Add(rand());
    }
    Sort(list, Greater<int>());
    Console.Out() << list << endl();
}
```

### Implementation

algorithm.cm, page 11

## 2.4.197 Sort(C&, R) Function

Sorts the elements of a forward container to order according to the given ordering relation.

### Syntax

```
public void Sort(C& c, R r);
```

### Constraint

C is ForwardContainer and R is Relation and R.Domain is C.Iterator.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| c | C& | A forward container. |
| r | R | An ordering relation. |

### Remarks

First the elements from the forward container are copied to a List<T>, then the List<T> is sorted, and finally the elements from the List<T> are copied back to the forward container.

### 2.4.198 Sort(I, I) Function

Sorts the elements of a sequence to increasing order.

#### Syntax

```
public void Sort(I begin, I end);
```

#### Constraint

I is RandomAccessIterator and I.ValueType is TotallyOrdered

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A random access iterator pointing to the beginning of a sequence. |
| end | I | A random access iterator pointing one past the end of a sequence. |

#### Implementation

algorithm.cm, page 12

### 2.4.199 Sort(I, I, R) Function

Sorts the elements of a sequence to order according to the given ordering relation.

#### Syntax

```
public void Sort(I begin, I end, R r);
```

#### Constraint

I is RandomAccessIterator and R is Relation and R.Domain is I.ValueType

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | A random access iterator pointing to the beginning of a sequence. |
| end | I | A random access iterator pointing one past the end of a sequence. |
| r | R | An ordering relation. |

#### Implementation

algorithm.cm, page 11

## 2.4.200 Swap(T&, T&) Function

Exchanges two values.

### Syntax

```
public void Swap(T& left, T& right);
```

### Constraint

T is MoveConstructible and T is MoveAssignable and T is Destructible

### Parameters

| Name | Type | Description |
|---|---|---|
| left | T& | The first value. |
| right | T& | The second value. |

### Remarks

The values are converted to rvalues using Rvalue(T&&) function and exchanged by moving them.

### Implementation

algorithm.cm, page 1

### 2.4.201 ThrowConversionException(const System.String<char>&) Function

Throws ConversionException with the given error message.

#### Syntax

```
public void ThrowConversionException(const System.String<char>& message);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| message | const System.String<char>& | An error message. |

### 2.4.202 ToHexString(System.uhuge) Function

Returns a 128-bit value converted to hexadecimal representation.

#### Syntax

```
public System.String<char> ToHexString(System.uhuge x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | System.uhuge | A 128-bit value to convert. |

#### Returns

System.String<char>

A hexadecimal string.

### 2.4.203 ToHexString(U) Function

Converts an unsigned integer value to hexadecimal string representation.

#### Syntax

```
public System.String<char> ToHexString(U x);
```

#### Constraint

byte and ExplicitlyConvertible<U, U>

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | U | An unsigned integer value. |

#### Returns

System.String<char>

Returns $x$ converted to hexadecimal string representation.

#### Implementation

convert.cm, page 4

### 2.4.204 ToHexString(byte) Function

Converts a **byte** to hexadecimal string representation.

#### Syntax

```
public System.String<char> ToHexString(byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A **byte**. |

#### Returns

System.String<char>

Returns *b* converted to hexadecimal string representation.

#### Implementation

convert.cm, page 4

### 2.4.205 ToHexString(uint) Function

Converts an **uint** to hexadecimal string representation.

#### Syntax

```
public System.String<char> ToHexString(uint u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | uint | An **uint**. |

#### Returns

System.String<char>

Returns *u* converted to hexadecimal string representation.

#### Implementation

convert.cm, page 4

### 2.4.206 ToHexString(ulong) Function

Converts an **ulong** to hexadecimal string representation.

#### Syntax

```
public System.String<char> ToHexString(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | An **ulong**. |

#### Returns

System.String<char>

Returns *u* converted to hexadecimal string representation.

#### Implementation

convert.cm, page 4

### 2.4.207 ToHexString(ushort) Function

Converts an **ushort** to hexadecimal string representation.

#### Syntax

```
public System.String<char> ToHexString(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | An **ushort**. |

#### Returns

System.String<char>

Returns *u* converted to hexadecimal string representation.

#### Implementation

convert.cm, page 4

### 2.4.208 ToLower(const System.String<char>&) Function

Converts the given UTF-8 string to lower case.

#### Syntax

```
public System.String<char> ToLower(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to convert. |

#### Returns

System.String<char>

Returns the string converted to lower case.

### 2.4.209 ToLower(const System.String<uchar>&) Function

Converts the given UTF-32 string to lower case.

#### Syntax

```
public System.String<uchar> ToLower(const System.String<uchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to convert. |

#### Returns

System.String<uchar>

Returns the string converted to lower case.

### 2.4.210 ToLower(const System.String<wchar>&) Function

Converts the given UTF-16 string to lower case.

#### Syntax

```
public System.String<wchar> ToLower(const System.String<wchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to convert. |

#### Returns

System.String<wchar>

Returns the string converted to lower case.

### 2.4.211 ToString(I) Function

Converts a signed integer value to string representation.

#### Syntax

```
public System.String<char> ToString(I x);
```

#### Constraint

byte and ExplicitlyConvertible<U, U>

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | I | A signed integer value. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 1

### 2.4.212 ToString(System.Date) Function

Converts a Date to a string representation.

#### Syntax

```
public System.String<char> ToString(System.Date date);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| date | System.Date | A date to convert. |

## Returns

System.String<char>

Returns a string YYYY-MM-DD where YYYY is year of date in four digits, MM is month of date in two digits and DD is day of date in two digits.

### 2.4.213 ToString(System.uhuge) Function

Converts a 128-bit value to string representation.

#### Syntax

```
public System.String<char> ToString(System.uhuge x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | System.uhuge | A 128-bit value. |

#### Returns

System.String<char>

Returns $x$ converted to string.

### 2.4.214 ToString(U) Function

Converts an unsigned integer value to string representation.

#### Syntax

```
public System.String<char> ToString(U x);
```

#### Constraint

byte and ExplicitlyConvertible<U, U>

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | U | An unsigned integer value. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.215 ToString(bool) Function

Converts a Boolean value to string representation.

#### Syntax

```
public System.String<char> ToString(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean value. |

#### Returns

System.String<char>

If *b* returns "true" else returns "false".

#### Implementation

convert.cm, page 3

### 2.4.216 ToString(byte) Function

Converts a **byte** to string representation.

#### Syntax

```
public System.String<char> ToString(byte x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | byte | A **byte**. |

#### Returns

System.String<char>

Returns a **byte** converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.217 ToString(char) Function

Converts a character to string representation.

#### Syntax

```
public System.String<char> ToString(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character. |

#### Returns

System.String<char>

Returns *c* converted to string representation.

#### Implementation

convert.cm, page 3

### 2.4.218 ToString(double) Function

Converts a **double** to string representation.

#### Syntax

```
public System.String<char> ToString(double x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | double | A **double**. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Remarks

The maximum number of decimal places in the conversion is 15.

#### Implementation

convert.cm, page 3

### 2.4.219 ToString(double, int) Function

Converts a **double** to string representation using the given maximum number of decimal places.

#### Syntax

```
public System.String<char> ToString(double x, int maxNumDecimals);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | double | A **double**. |
| maxNumDecimals | int | Maximum number of decimal digits. |

#### Returns

System.String<char>

Returns $x$ converted to string representation using the given maximum number of decimal places.

#### Implementation

convert.cm, page 3

### 2.4.220 ToString(double, int, int) Function

Converts a **double** to string representation using the given minimum and maximum number of decimal places.

#### Syntax

```
public System.String<char> ToString(double x, int minNumDecimals, int maxNumDecimals);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | double | A **double**. |
| minNumDecimals | int | Minimum number of decimal digits. |
| maxNumDecimals | int | Maximum number of decimal digits. |

## Returns

System.String<char>

Returns $x$ converted to string representation using the given minimum and maximum number of decimal places.

### 2.4.221 ToString(int) Function

Converts an **int** to string representation.

#### Syntax

```
public System.String<char> ToString(int x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | int | An **int**. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.222 ToString(long) Function

Converts a **long** to string representation.

#### Syntax

```
public System.String<char> ToString(long x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | long | A **long**. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.223 ToString(sbyte) Function

Converts an **sbyte** to string representation.

#### Syntax

```
public System.String<char> ToString(sbyte x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | sbyte | An **sbyte**. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.224 ToString(short) Function

Converts a **short** to string representation.

#### Syntax

```
public System.String<char> ToString(short x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | short | An **short**. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.225 ToString(uchar) Function

Converts the given UTF-32 character to a UTF-8 string representation.

#### Syntax

```
public System.String<char> ToString(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A character to convert. |

#### Returns

System.String<char>

Returns *c* converted to string representation.

### 2.4.226 ToString(uint) Function

Converts an **uint** to string representation.

#### Syntax

```
public System.String<char> ToString(uint x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | uint | An **uint**. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.227 ToString(ulong) Function

Converts an **ulong** to string representation.

#### Syntax

```
public System.String<char> ToString(ulong x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | ulong | An **ulong**. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.228 ToString(ushort) Function

Converts an **ushort** to string representation.

#### Syntax

```
public System.String<char> ToString(ushort x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | ushort | An **ushort**. |

#### Returns

System.String<char>

Returns $x$ converted to string representation.

#### Implementation

convert.cm, page 2

### 2.4.229 ToString(wchar) Function

Converts the given UTF-16 character to a UTF-8 string representation.

#### Syntax

```
public System.String<char> ToString(wchar c);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A character to convert. |

## Returns

System.String<char>

Returns *c* converted to string representation.

### 2.4.230 ToUpper(const System.String<char>&) Function

Converts the given UTF-8 string to upper case.

#### Syntax

```
public System.String<char> ToUpper(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to convert. |

#### Returns

System.String<char>

Returns the string converted to upper case.

### 2.4.231 ToUpper(const System.String<uchar>&) Function

Converts the given UTF-32 string to upper case.

#### Syntax

```
public System.String<uchar> ToUpper(const System.String<uchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to convert. |

#### Returns

System.String<uchar>

Returns the string converted to upper case.

### 2.4.232 ToUpper(const System.String<wchar>&) Function

Converts the given UTF-16 string to upper case.

#### Syntax

```
public System.String<wchar> ToUpper(const System.String<wchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to convert. |

#### Returns

System.String<wchar>

Returns the string converted to upper case.

## 2.4.233 Transform(I, I, O, F) Function

Transforms an input sequence to an output sequence using a unary function.

### Syntax

```
public O Transform(I begin, I end, O to, F fun);
```

### Constraint

I is InputIterator and O is OutputIterator and F is UnaryFunction and F.ArgumentType is I.ValueType and CopyAssignable<O.ValueType, F.ResultType>

### Parameters

| Name | Type | Description |
|---|---|---|
| begin | I | An input iterator pointing to the beginning of an input sequence. |
| end | I | An input iterator pointing one past the end of an input sequence. |
| to | O | An output iterator pointing to the beginning of an output sequence. |
| fun | F | A unary function object. |

### Returns

O

Returns an output iterator pointing one past the end of the output sequence.

#### 2.4.233.1 Example

```
using System;
using System.Collections;

//  Writes:
//  2, 4, 6

public class Double<A>: UnaryFun<A, A> where A is AdditiveSemigroup
{
    public A operator()(const A& x) const
    {
        return x + x;
    }
}

void main()
{
    List<int> list;
    list.Add(1);
    list.Add(2);
    list.Add(3);
```

```
    List<int> doubledList;
    Transform(list.CBegin(), list.CEnd(), BackInserter(doubledList), Double<
        int>());
    Console.Out() << doubledList << endl();
}
```

## Implementation

algorithm.cm, page 8

## 2.4.234 Transform(I1, I1, I2, O, F) Function

Transforms two input sequences to an ouput sequence using a binary function.

### Syntax

```
public O Transform(I1 begin1, I1 end1, I2 begin2, O to, F fun);
```

### Constraint

I1 is InputIterator and I2 is InputIterator and O is OutputIterator and F is BinaryFunction and F.FirstArgumentType is I1.ValueType and F.SecondArgumentType is I2.ValueType and CopyAssignable<O.ValueType, F.ResultType>

### Parameters

| Name | Type | Description |
|---|---|---|
| begin1 | I1 | An input iterator pointing to the beginning of the first input sequence. |
| end1 | I1 | An input iterator pointing one past the end of the first input sequence. |
| begin2 | I2 | An input iterator pointing to the beginning of the second input sequence. |
| to | O | An output iterator pointing to the beginning of the output sequence. |
| fun | F | A binary function object. |

### Returns

O

Returns an output iterator pointing one past the end of the output sequence.

#### 2.4.234.1 Example

```
using System;
using System.Collections;

//  Writes:
//  4, 6, 8

void main()
{
    List<int> list1;
    list1.Add(1);
    list1.Add(2);
    list1.Add(3);
    List<int> list2;
    list2.Add(3);
    list2.Add(4);
```

```
    list2.Add(5);
    List<int> sum;
    Transform(list1.CBegin(), list1.CEnd(), list2.CBegin(), BackInserter(sum
        ), Plus<int>());
    Console.Out() << sum << endl();
}
```

## Implementation

algorithm.cm, page 8

## 2.4.235 UpperBound(I, I, const T&) Function

Finds a position of the first element in a sorted sequence that is greater than the given value.

### Syntax

```
public I UpperBound(I first, I last, const T& value);
```

### Constraint

I is ForwardIterator and TotallyOrdered<T, I.ValueType>

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator pointing to the beginning of a sorted sequence. |
| last | I | A forward iterator pointing one past the end of a sorted sequence. |
| value | const T& | A value to search. |

### Returns

I

Returns an iterator pointing to the first element in a sorted sequence that is greater than the given value, if there is one, otherwise returns *last*.

### 2.4.235.1 Example

```
using System;
using System.Collections;

//  Writes:
//  1, 1, 2, 2, 3, 3
//  upper bound of 2 is 3 at position 4

void main()
{
    List<int> list;
    list.Add(1);
    list.Add(2);
    list.Add(3);
    list.Add(2);
    list.Add(3);
    list.Add(1);
    Sort(list);
    Console.Out() << list << endl();
    List<int>.ConstIterator ub = UpperBound(list.CBegin(), list.CEnd(), 2);
    if (ub != list.CEnd())
    {
        Console.Out() << "upper bound of 2 is " << *ub << " at position " <<
            Distance(list.CBegin(), ub) << endl();
```

```
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

## Implementation

algorithm.cm, page 4

## 2.4.236 UpperBound(I, I, const T&, R) Function

Finds a position of the first element in a sorted sequence that is greater than the given value according to the given ordering relation.

### Syntax

```
public I UpperBound(I first, I last, const T& value, R r);
```

### Constraint

I is ForwardIterator and T is I.ValueType and R is Relation and R.Domain is I.ValueType

### Parameters

| Name | Type | Description |
|---|---|---|
| first | I | A forward iterator pointing to the beginning of a sorted sequence. |
| last | I | A forward iterator pointing one past the end of a sorted sequence. |
| value | const T& | A value to search. |
| r | R | An ordering relation. |

### Returns

I

Returns an iterator pointing to the first element in a sorted sequence that is greater than the given value according to the given ordering relation, if there is one, otherwise returns *last*.

### Implementation

algorithm.cm, page 5

### 2.4.237 endl() Function

Returns EndLine object that represents an end of line character.

#### Syntax

```
public System.EndLine endl();
```

#### Returns

System.EndLine

Returns EndLine object that represents and end of line character.

## 2.5 Enumerations

| Enumeration | Description |
|---|---|
| CharClass | A character classification enumeration. |

### 2.5.237.1 CharClass Enumeration

A character classification enumeration.

**Enumeration Constants**

| Constant | Value | Description |
|---|---|---|
| alnum | 7 | Alphanumeric character. |
| alpha | 3 | Alphabetic character. |
| cntrl | 16 | Control character. |
| digit | 4 | Digit character. |
| graph | 32 | Graphical character. |
| lower | 1 | Lower case alphabetic character. |
| none | 0 | No classification. |
| print | 64 | Printable character. |
| punct | 128 | Punctuation character. |
| space | 256 | White space character. |
| upper | 2 | Upper case alphabetic character. |
| xdigit | 8 | Hexadecimal digit character. |

## 2.6 Constants

| Constant | Type | Value | Description |
|---|---|---|---|
| EXIT_CHAR_-CLASS_TABLE_-ALLOCATE | int | 249 | Program exit status when the character class table could not be allocated. |
| InsertionSortThreshold | int | 16 | A threshold value for doing insertion sort. |

# 3 System.Collections Namespace

Contains collection classes and functions that operate on collections.

Figure 3.1 contains the classes in this namespace.

Figure 3.1: Class Diagram: Collection Classes

System.Collections.BitSet

System.Collections.ForwardList<T>

System.Collections.ForwardListNodeIterator<T, R, P>

System.Collections.List<T>

System.Collections.Map<Key, Value, KeyCompare>

System.Collections.Queue<T>

System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>

System.Collections.RedBlackTreeNodeIterator<T, R, P>

System.Collections.Set<T, C>

System.Collections.Stack<T>

## 3.7 Classes

| Class | Description |
|---|---|
| BitSet | A compact sequence of bits. |
| Bucket<T> | A hash table bucket. |
| ForwardList<T> | A singly linked list of elements. |
| ForwardListNode<T> | A forward list node type. |
| ForwardListNodeIterator<T, R, P> | A forward iterator that iterates throught a ForwardList<T>. |
| HashMap<K, T, H, C> | An associative container of key-value pairs organized in a hash table. The keys need not be ordered. |
| HashSet<T, H, C> | A set of unique elements organized in a hash table. The elements need not be ordered. |
| Hasher<T> | Default hash function. |
| Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare> | A hash table of unique elements used to implement HashMap<K, T, H, C> and HashSet<T, H, C>. The keys of elements need not be ordered. |
| HashtableBase<T> | Implementation detail. |
| HashtableIterator<T, R, P> | A hash table iterator type. |
| LinkedList<T> | A doubly linked list class. |
| LinkedListBase | Implementation detail. |
| LinkedListNode<T> | A linked list node type. |
| LinkedListNodeBase | Implementation detail. |
| LinkedListNodeIterator<T, R, P> | A linked list iterator type. |
| List<T> | A container of elements in which the contained elements are in consecutive locations in memory. |
| Map<Key, Value, KeyCompare> | An associative container of key-value pairs organized in a red-black tree. The keys need to be ordered. |

| | |
|---|---|
| Queue<T> | A first-in first-out data structure. |
| RedBlackTree<KeyType, ValueType, KeyOf-Value, Compare> | A self-balancing binary search tree of unique elements used to implement Set<T, C> and Map<-Key, Value, KeyCompare>. The keys of the elements in the tree need to be ordered. |
| RedBlackTreeNodeIterator<T, R, P> | A bidirectional iterator that iterates through the elements in a reb-black tree. |
| Set<T, C> | A container that contains a set of unique elements organized in a red-black tree. The elements need to be ordered. |
| Stack<T> | A last-in first-out data structure. |

### 3.7.1 BitSet Class

A compact sequence of bits.

#### Syntax

```
public class BitSet;
```

#### 3.7.1.1 Example

```
using System;
using System.Collections;

//  Writes:
//  11111111
//  all bits are 1
//  10101010
//  s[0] == 1
//  01010101
//  now s[0] == 0

void main()
{
    int n = 8;
    BitSet s(n);
    s.Set();
    Console.Out() << s.ToString() << endl();
    if (s.All())
    {
        Console.Out() << "all bits are 1" << endl();
    }
    for (int i = 0; i < n; ++i)
    {
        if (i % 2 == 1)
        {
            s.Reset(i);
        }
    }
    Console.Out() << s.ToString() << endl();
    if (s[0])
    {
        Console.Out() << "s[0] == 1" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
    s.Flip();
    Console.Out() << s.ToString() << endl();
    if (!s[0])
    {
        Console.Out() << "now s[0] == 0" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
```

```
}
```

#### 3.7.1.2 Member Functions

| Member Function | Description |
|---|---|
| BitSet() | Constructor. Constructs an empty bit set. |
| BitSet(const System.Collections.BitSet&) | Copy constructor. |
| operator=(const System.Collections.BitSet&) | Copy assignment. |
| BitSet(System.Collections.BitSet&&) | Move constructor. |
| operator=(System.Collections.BitSet&&) | Move assignment. |
| All() const | Returns true if all the bits are 1, false otherwise. |
| Any() const | Returns true if any bit is 1, false otherwise. |
| BitSet(const System.String<char>&) | Constructor. Constructs a bit set containing given bits. |
| BitSet(int) | Constructor. Constructs a bit set capable of holding given number of bits. |
| Clear() | Makes the bit set empty. |
| Count() const | Returns the number of bits the bit set contains. |
| Flip() | Toggles the values of the bits that the bit set contains. |
| Flip(int) | Toggles the value of the bit with the given index. |
| None() const | Returns true if all the bits are 0, false otherwise. |
| Reset() | Resets all the bits to 0. |
| Reset(int) | Resets the bit with the given index to 0. |
| Resize(int) | Resizes the bit set to contain the given number of bits. |
| Set() | Sets all the bits to 1. |
| Set(int) | Sets the bit with the given index to 1. |
| Set(int, bool) | Sets the bit with the given index to the given value. |

| | |
|---|---|
| Test(int) const | Returns true if a bit with the given index is 1, false otherwise. |
| ToString() const | Returns the string representation of the bit set. |
| operator==(const System.Collections.BitSet&) const | Compares this bit set and given bit set for equality. |
| operator[](int) const | Returns true if the bit with the given index is 1 and false if it is 0. |
| ~BitSet() | Destructor. |

### BitSet() Member Function

Constructor. Constructs an empty bit set.

#### Syntax

```
public BitSet();
```

## Implementation

bitset.cm, page 1

### BitSet(const System.Collections.BitSet&) Member Function

Copy constructor.

#### Syntax

```
public BitSet(const System.Collections.BitSet& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | const System.Collections.BitSet& | |

### operator=(const System.Collections.BitSet&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.BitSet& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | const System.Collections.BitSet& | |

### BitSet(System.Collections.BitSet&&) Member Function

Move constructor.

#### Syntax

```
public BitSet(System.Collections.BitSet&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.Collections.BitSet&& | |

### operator=(System.Collections.BitSet&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.BitSet&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.Collections.BitSet&& | |

### All() const Member Function

Returns true if all the bits are 1, false otherwise.

#### Syntax

```
public bool All() const;
```

#### Returns

bool

Returns true if all the bits are 1, false otherwise.

#### Implementation

bitset.cm, page 3

### Any() const Member Function

Returns true if any bit is 1, false otherwise.

### Syntax

```
public bool Any() const;
```

### Returns

bool

Returns true if any bit is 1, false otherwise.

### Implementation

bitset.cm, page 4

### BitSet(const System.String<char>&) Member Function

Constructor. Constructs a bit set containing given bits.

#### Syntax

```
public BitSet(const System.String<char>& bits_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| bits_ | const System.String<char>& | Bit string. |

### BitSet(int) Member Function

Constructor. Constructs a bit set capable of holding given number of bits.

#### Syntax

```
public BitSet(int numBits_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| numBits_ | int | Number of bits the bit set will contain. |

#### Implementation

bitset.cm, page 1

### Clear() Member Function

Makes the bit set empty.

### Syntax

```
public void Clear();
```

### Implementation

bitset.cm, page 2

### Count() const Member Function

Returns the number of bits the bit set contains.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of bits the bit set contains.

#### Implementation

bitset.cm, page 2

### Flip() Member Function

Toggles the values of the bits that the bit set contains.

### Syntax

```
public void Flip();
```

### Implementation

bitset.cm, page 3

### Flip(int) Member Function

Toggles the value of the bit with the given index.

### Syntax

```
public void Flip(int pos);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| pos | int | Index of the bit to toggle. |

### Implementation

bitset.cm, page 3

### None() const Member Function

Returns true if all the bits are 0, false otherwise.

#### Syntax

```
public bool None() const;
```

#### Returns

bool

Returns true if all the bits are 0, false otherwise.

#### Implementation

bitset.cm, page 4

### Reset() Member Function

Resets all the bits to 0.

### Syntax

```
public void Reset();
```

### Implementation

bitset.cm, page 2

### Reset(int) Member Function

Resets the bit with the given index to 0.

### Syntax

```
public void Reset(int pos);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| pos | int | Index of the bit to reset. |

### Implementation

bitset.cm, page 2

### Resize(int) Member Function

Resizes the bit set to contain the given number of bits.

### Syntax

```
public void Resize(int numBits_);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| numBits_ | int | Number of bits the bit set will contain. |

### Implementation

bitset.cm, page 2

### Set() Member Function

Sets all the bits to 1.

### Syntax

```
public void Set();
```

### Implementation

bitset.cm, page 2

### Set(int) Member Function

Sets the bit with the given index to 1.

### Syntax

```
public void Set(int pos);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| pos | int | Index of the bit to set. |

### Implementation

bitset.cm, page 2

### Set(int, bool) Member Function

Sets the bit with the given index to the given value.

### Syntax

```
public void Set(int pos, bool bit);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| pos | int | Index of the bit to set. |
| bit | bool | If true, the bit will be set to 1, otherwise it will be reset to 0. |

### Implementation

bitset.cm, page 3

### Test(int) const Member Function

Returns true if a bit with the given index is 1, false otherwise.

### Syntax

```
public bool Test(int pos) const;
```

### Parameters

| Name | Type | Description |
|---|---|---|
| pos | int | Index of the bit to test. |

### Returns

bool

Returns true if a bit with the given index is 1, false otherwise.

### Implementation

bitset.cm, page 3

### ToString() const Member Function

Returns the string representation of the bit set.

#### Syntax

```
public System.String<char> ToString() const;
```

#### Returns

System.String<char>

Returns the string representation of the bit set.

#### Implementation

bitset.cm, page 5

### operator==(const System.Collections.BitSet&) const Member Function

Compares this bit set and given bit set for equality.

#### Syntax

```
public bool operator==(const System.Collections.BitSet& that) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.BitSet& | A bit set to compare with. |

#### Returns

bool

Returns true if both bitsets contain the same number of equal bits, false otherwise.

#### Implementation

bitset.cm, page 4

### operator[](int) const Member Function

Returns true if the bit with the given index is 1 and false if it is 0.

#### Syntax

```
public bool operator[](int index) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| index | int | Index of the bit to test. |

#### Returns

bool

Returns true if the bit with the given index is 1 and false if it is 0.

#### Implementation

bitset.cm, page 3

### ~BitSet() Member Function

Destructor.

#### Syntax

```
public ~BitSet();
```

### 3.7.2 Bucket<T> Class

A hash table bucket.

#### Syntax

```
public class Bucket<T>;
```

#### Constraint

T is Semiregular

#### 3.7.2.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| ValueType | T | Type of value contained by the bucket. |

#### 3.7.2.2 Member Functions

| Member Function | Description |
|---|---|
| Bucket<T>() | Default constructor. |
| Bucket<T>(const System.Collections.Bucket<-T>&) | Copy constructor. |
| operator=(const System.Collections.Bucket<T>-&) | Copy assignment. |
| Bucket<T>(System.Collections.Bucket<T>&&) | Move constructor. |
| operator=(System.Collections.Bucket<T>&&) | Move assignment. |
| Bucket<T>(const T&, System.Collections.-Bucket<T>*) | Constructor. Initializes the bucket with a pointer to the next bucket and a value. |
| Next() const | Returns a pointer to the next bucket. |
| SetNext(System.Collections.Bucket<T>*) | Sets the pointer to the next bucket. |
| Value() const | Returns a constant reference to the value contained by the bucket. |

### Bucket<T>() Member Function

Default constructor.

#### Syntax

```
public Bucket<T>();
```

### Bucket<T>(const System.Collections.Bucket<T>&) Member Function

Copy constructor.

#### Syntax

```
public Bucket<T>(const System.Collections.Bucket<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Bucket<T>& | Argument to copy. |

### operator=(const System.Collections.Bucket<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.Bucket<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Bucket<T>& | Argument to assign. |

### Bucket<T>(System.Collections.Bucket<T>&&) Member Function

Move constructor.

#### Syntax

```
public Bucket<T>(System.Collections.Bucket<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Bucket<T>&& | Argument to move from. |

### operator=(System.Collections.Bucket<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.Bucket<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Bucket<T>&& | Argument to assign from. |

### Bucket<T>(const T&, System.Collections.Bucket<T>*) Member Function

Constructor. Initializes the bucket with a pointer to the next bucket and a value.

#### Syntax

```
public Bucket<T>(const T& value_, System.Collections.Bucket<T>* next_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value_ | const T& | A value contained by the bucket. |
| next_ | System.Collections.Bucket<T>* | Pointer to the next bucket. |

### Next() const Member Function

Returns a pointer to the next bucket.

#### Syntax

```
public System.Collections.Bucket<T>* Next() const;
```

#### Returns

System.Collections.Bucket<T>*

Returns a pointer to the next bucket.

### SetNext(System.Collections.Bucket<T>*) Member Function

Sets the pointer to the next bucket.

#### Syntax

```
public void SetNext(System.Collections.Bucket<T>* next_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| next_ | System.Collections.Bucket<T>* | A pointer to the next bucket. |

### Value() const Member Function

Returns a constant reference to the value contained by the bucket.

### Syntax

```
public const T& Value() const;
```

### Returns

const T&

Returns a constant reference to the value contained by the bucket.

## 3.7.3 ForwardList<T> Class

A singly linked list of elements.

### Syntax

```
public class ForwardList<T>;
```

### Constraint

T is Regular

### Model of

ForwardContainer<T>

#### 3.7.3.1 Remarks

**ForwardList<T>** is suitable only for very short sequences of items. Often a List<T> is more appropriate.

#### 3.7.3.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| ConstIterator | System.Collections.-ForwardListNodeIterator<T, const T&, const T*> | A constant iterator type. |
| Iterator | System.Collections.-ForwardListNodeIterator<T, T&, T*> | An iterator type. |
| ValueType | T | The type of the contained element. |

#### 3.7.3.3 Member Functions

| Member Function | Description |
|---|---|
| ForwardList<T>() | Constructor. Constructs an empty forward list. |
| ForwardList<T>(const System.Collections.-ForwardList<T>&) | Copy constructor. |
| operator=(const System.Collections.-ForwardList<T>&) | Copy assignment. |
| ForwardList<T>(System.Collections.-ForwardList<T>&&) | Move constructor. |
| operator=(System.Collections.ForwardList<T>-&&) | Move assignment. |
| Begin() | Returns an iterator pointing to the first element of the forward list, or End() if the forward list is empty. |
| Begin() const | Returns a constant iterator pointing to the first element of the forward list, or CEnd() const if the forward list is empty. |
| CBegin() const | Returns a constant iterator pointing to the first element of the forward list, or CEnd() const if the forward list is empty. |
| CEnd() const | Returns a constant iterator pointing to one past the end of the forward list. |
| Clear() | Makes the forward list empty. |
| Count() const | Counts the number of elements in the forward list. |
| End() | Returns an iterator pointing to one past the end of the forward list. |
| End() const | Returns a constant iterator pointing to one past the end of the forward list. |

| | |
|---|---|
| Front() const | Returns a constant reference to the first element in the forward list. |
| InsertAfter(System.Collections.-ForwardListNodeIterator<T, T&, T*>, const T&) | Inserts an element after the position pointed by the given iterator, or to the front of the forward list if the forward list is empty and the iterator is an End() iterator. |
| InsertFront(const T&) | Inserts an element to the front of the forward list. |
| IsEmpty() const | Returns true if the forward list is empty, false otherwise. |
| Remove(const T&) | Removes all occurrences of the given value from the forward list. |
| RemoveAfter(System.Collections.-ForwardListNodeIterator<T, T&, T*>) | Removes an element after the position pointed by the given iterator. |
| RemoveFront() | Removes an element from the front of the forward list. |
| ~ForwardList<T>() | Destructor. |

### ForwardList<T>() Member Function

Constructor. Constructs an empty forward list.

#### Syntax

```
public ForwardList<T>();
```

## Implementation

fwdlist.cm, page 3

### ForwardList<T>(const System.Collections.ForwardList<T>&) Member Function

Copy constructor.

#### Syntax

```
public ForwardList<T>(const System.Collections.ForwardList<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.ForwardList<T>& | A forward list to copy. |

## Implementation

fwdlist.cm, page 3

### operator=(const System.Collections.ForwardList<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.ForwardList<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.ForwardList<T>& | A forward list to assign. |

## Implementation

fwdlist.cm, page 3

### ForwardList<T>(System.Collections.ForwardList<T>&&) Member Function

Move constructor.

#### Syntax

```
public ForwardList<T>(System.Collections.ForwardList<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.ForwardList<T>&& | A forward list to move from. |

## Implementation

fwdlist.cm, page 3

### operator=(System.Collections.ForwardList<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.ForwardList<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.ForwardList<T>&& | A forward list to move from. |

## Implementation

fwdlist.cm, page 3

### Begin() Member Function

Returns an iterator pointing to the first element of the forward list, or End() if the forward list is empty.

### Syntax

```
public System.Collections.ForwardListNodeIterator<T, T&, T*> Begin();
```

### Returns

System.Collections.ForwardListNodeIterator<T, T&, T*>

Returns an iterator pointing to the first element of the forward list, or End() if the forward list is empty.

### Implementation

fwdlist.cm, page 4

### Begin() const Member Function

Returns a constant iterator pointing to the first element of the forward list, or CEnd() const if the forward list is empty.

### Syntax

```
public System.Collections.ForwardListNodeIterator<T, const T&, const T*> Begin()
const;
```

### Returns

System.Collections.ForwardListNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to the first element of the forward list, or CEnd() const if the forward list is empty.

### Implementation

fwdlist.cm, page 4

### CBegin() const Member Function

Returns a constant iterator pointing to the first element of the forward list, or CEnd() const if the forward list is empty.

### Syntax

```
public System.Collections.ForwardListNodeIterator<T, const T&, const T*> CBegin()
const;
```

### Returns

System.Collections.ForwardListNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to the first element of the forward list, or CEnd() const if the forward list is empty.

### Implementation

fwdlist.cm, page 4

### CEnd() const Member Function

Returns a constant iterator pointing to one past the end of the forward list.

#### Syntax

```
public System.Collections.ForwardListNodeIterator<T, const T&, const T*> CEnd() const;
```

#### Returns

System.Collections.ForwardListNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to one past the end of the forward list.

#### Implementation

fwdlist.cm, page 4

### Clear() Member Function

Makes the forward list empty.

### Syntax

```
public void Clear();
```

### Implementation

fwdlist.cm, page 3

### Count() const Member Function

Counts the number of elements in the forward list.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the forward list.

#### Remarks

This is a *O(n)* operation where $n$ is the number of elements in the forward list.

#### Implementation

fwdlist.cm, page 3

### End() Member Function

Returns an iterator pointing to one past the end of the forward list.

### Syntax

```
public System.Collections.ForwardListNodeIterator<T, T&, T*> End();
```

### Returns

System.Collections.ForwardListNodeIterator<T, T&, T*>

Returns an iterator pointing to one past the end of the forward list.

### Implementation

fwdlist.cm, page 4

### End() const Member Function

Returns a constant iterator pointing to one past the end of the forward list.

### Syntax

```
public System.Collections.ForwardListNodeIterator<T, const T&, const T*> End() const;
```

### Returns

System.Collections.ForwardListNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to one past the end of the forward list.

### Implementation

fwdlist.cm, page 4

### Front() const Member Function

Returns a constant reference to the first element in the forward list.

### Syntax

```
public const T& Front() const;
```

### Returns

const T&

Returns a constant reference to the first element in the forward list.

### Implementation

fwdlist.cm, page 4

### InsertAfter(System.Collections.ForwardListNodeIterator<T, T&, T*>, const T&) Member Function

Inserts an element after the position pointed by the given iterator, or to the front of the forward list if the forward list is empty and the iterator is an End() iterator.

#### Syntax

```
public System.Collections.ForwardListNodeIterator<T, T&, T*> InsertAfter(System.Collections.Forw
T&, T*> pos, const T& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.ForwardListNodeIterator<T, T&, T*> | An iterator. |
| value | const T& | A value to insert. |

#### Returns

System.Collections.ForwardListNodeIterator<T, T&, T*>

Returns an iterator pointing to the inserted element.

#### Implementation

fwdlist.cm, page 4

### InsertFront(const T&) Member Function

Inserts an element to the front of the forward list.

#### Syntax

```
public System.Collections.ForwardListNodeIterator<T, T&, T*> InsertFront(const T&
value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const T& | A value to insert. |

#### Returns

System.Collections.ForwardListNodeIterator<T, T&, T*>

Returns an iterator pointing to the inserted element.

#### Implementation

fwdlist.cm, page 4

### IsEmpty() const Member Function

Returns true if the forward list is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the forward list is empty, false otherwise.

#### Implementation

fwdlist.cm, page 3

### Remove(const T&) Member Function

Removes all occurrences of the given value from the forward list.

#### Syntax

```
public void Remove(const T& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const T& | A value to remove. |

#### Implementation

fwdlist.cm, page 5

### RemoveAfter(System.Collections.ForwardListNodeIterator<T, T&, T*>) Member Function

Removes an element after the position pointed by the given iterator.

#### Syntax

```
public void RemoveAfter(System.Collections.ForwardListNodeIterator<T, T&, T*> pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.ForwardListNodeIterator<T, T&, T*> | An iterator. |

#### Implementation

fwdlist.cm, page 5

### RemoveFront() Member Function

Removes an element from the front of the forward list.

### Syntax

```
public void RemoveFront();
```

### Implementation

fwdlist.cm, page 4

#### ~ForwardList<T>() Member Function

Destructor.

#### Syntax

```
public ~ForwardList<T>();
```

#### Implementation

fwdlist.cm, page 3

### 3.7.3.4 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.Collections.ForwardList<T>&, const System.Collections.ForwardList<T>&) | Compares two forward lists for less than relationship and returns true if the first forward list comes lexicographically before the second forward list, false otherwise. |
| operator==(const System.Collections.ForwardList<T>&, const System.Collections.ForwardList<T>&) | Compares two forward lists for equality and returns true if both contain the same number of pairwise equal elements, false otherwise. |

### operator<(const System.Collections.ForwardList<T>&, const System.Collections.ForwardList<T>&) Function

Compares two forward lists for less than relationship and returns true if the first forward list comes lexicographically before the second forward list, false otherwise.

#### Syntax

```
public bool operator<(const System.Collections.ForwardList<T>& left, const System.Collections.F
right);
```

## Constraint

T is TotallyOrdered

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.ForwardList<T>& | The first forward list. |
| right | const System.Collections.ForwardList<T>& | The second forward list. |

#### Returns

bool

Returns true if the first forward list comes lexicographically before the second forward list, false otherwise.

#### Implementation

fwdlist.cm, page 6

**operator==(const System.Collections.ForwardList<T>&, const System.Collections.ForwardList<T>&
Function**

Compares two forward lists for equality and returns true if both contain the same number of pairwise equal elements, false otherwise.

### Syntax

```
public bool operator==(const System.Collections.ForwardList<T>& left, const System.Collections.F
right);
```

## Constraint

T is Regular

### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.ForwardList<T>& | The first forward list. |
| right | const System.Collections.ForwardList<T>& | The second forward list. |

### Returns

bool

returns true if both contain the same number of pairwise equal elements, false otherwise.

### Implementation

fwdlist.cm, page 6

## 3.7.4 ForwardListNode<T> Class

A forward list node type.

### Syntax

```
public class ForwardListNode<T>;
```

### Constraint

T is Semiregular

#### 3.7.4.1 Member Functions

| Member Function | Description |
|---|---|
| ForwardListNode<T>() | Default constructor. |
| ForwardListNode<T>(const System.Collections.ForwardListNode<T>&) | Copy constructor. |
| operator=(const System.Collections.ForwardListNode<T>&) | Copy assignment. |
| ForwardListNode<T>(System.Collections.ForwardListNode<T>&&) | Move constructor. |
| operator=(System.Collections.ForwardListNode<T>&&) | Move assignment. |
| ForwardListNode<T>(System.Collections.ForwardListNode<T>*, const T&) | Constructor. Initializes the forward list node with a pointer to the next forward list node and a value. |
| Next() const | Returns a pointer to the next forward list node. |
| SetNext(System.Collections.ForwardListNode<T>*) | Sets the pointer to the next forward list node. |
| Value() | Returns a reference to the value contained by the forward list node. |
| Value() const | Returns a constant reference to the value contained by the forward list node. |

### ForwardListNode<T>() Member Function

Default constructor.

#### Syntax

```
public ForwardListNode<T>();
```

### ForwardListNode<T>(const System.Collections.ForwardListNode<T>&) Member Function

Copy constructor.

#### Syntax

```
public ForwardListNode<T>(const System.Collections.ForwardListNode<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.ForwardListNode<T>& | Argument to copy. |

### operator=(const System.Collections.ForwardListNode<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.ForwardListNode<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.ForwardListNode<T>& | Argument to assign. |

### ForwardListNode<T>(System.Collections.ForwardListNode<T>&&) Member Function

Move constructor.

#### Syntax

```
public ForwardListNode<T>(System.Collections.ForwardListNode<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.ForwardListNode<T>&& | Argument to move from. |

### operator=(System.Collections.ForwardListNode<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.ForwardListNode<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.ForwardListNode<T>&& | Argument to assign from. |

### ForwardListNode<T>(System.Collections.ForwardListNode<T>*, const T&) Member Function

Constructor. Initializes the forward list node with a pointer to the next forward list node and a value.

#### Syntax

```
public ForwardListNode<T>(System.Collections.ForwardListNode<T>* next_, const T&
value_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| next_ | System.Collections.ForwardListNode<T>* | A pointer to the next forward list node. |
| value_ | const T& | A value. |

### Next() const Member Function

Returns a pointer to the next forward list node.

#### Syntax

```
public System.Collections.ForwardListNode<T>* Next() const;
```

#### Returns

System.Collections.ForwardListNode<T>*

Returns a pointer to the next forward list node.

### SetNext(System.Collections.ForwardListNode<T>*) Member Function

Sets the pointer to the next forward list node.

#### Syntax

```
public void SetNext(System.Collections.ForwardListNode<T>* next_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| next_ | System.Collections.ForwardListNode<T>* | A pointer to the next forward list node. |

### Value() Member Function

Returns a reference to the value contained by the forward list node.

### Syntax

```
public T& Value();
```

### Returns

T&

Returns a reference to the value contained by the forward list node.

### Value() const Member Function

Returns a constant reference to the value contained by the forward list node.

### Syntax

```
public const T& Value() const;
```

### Returns

const T&

Returns a constant reference to the value contained by the forward list node.

### 3.7.5 ForwardListNodeIterator<T, R, P> Class

A forward iterator that iterates throught a ForwardList<T>.

#### Syntax

```
public class ForwardListNodeIterator<T, R, P>;
```

#### Model of

ForwardIterator<T>

#### 3.7.5.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| PointerType | P | The type of a pointer to an element. |
| ReferenceType | R | The type of a reference to an element. |
| ValueType | T | The type of an element. |

#### 3.7.5.2 Member Functions

| Member Function | Description |
|---|---|
| ForwardListNodeIterator<T, R, P>() | Constructor. Default constructs a forward list node iterator. |
| ForwardListNodeIterator<T, R, P>(const System.Collections.ForwardListNodeIterator<T, R, P>&) | Copy constructor. |
| operator=(const System.Collections.-ForwardListNodeIterator<T, R, P>&) | Copy assignment. |
| ForwardListNodeIterator<T, R, P>(System.-Collections.ForwardListNodeIterator<T, R, P>&&) | Move constructor. |
| operator=(System.Collections.-ForwardListNodeIterator<T, R, P>&&) | Move assignment. |
| ForwardListNodeIterator<T, R, P>(System.-Collections.ForwardListNode<T>*) | Constructor. Constructs a forward list node iterator pointing to a forward list node. |
| GetNode() const | Returns the contained pointer to an element. |
| operator*() const | Returns a reference to an element. |
| operator++() | Advances the iterator pointing to the next element in the forward list. |
| operator->() const | Returns a pointer to an element. |

### ForwardListNodeIterator<T, R, P>() Member Function

Constructor. Default constructs a forward list node iterator.

#### Syntax

```
public ForwardListNodeIterator<T, R, P>();
```

## Implementation

fwdlist.cm, page 2

### ForwardListNodeIterator<T, R, P>(const System.Collections.ForwardListNodeIterator<T, R, P>&) Member Function

Copy constructor.

#### Syntax

```
public ForwardListNodeIterator<T, R, P>(const System.Collections.ForwardListNodeIterator<T,
R, P>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.ForwardListNodeIterator<T, R, P>& | Argument to copy. |

### operator=(const System.Collections.ForwardListNodeIterator<T, R, P>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.ForwardListNodeIterator<T, R, P>&
that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.ForwardListNodeIterator<T, R, P>& | Argument to assign. |

### ForwardListNodeIterator<T, R, P>(System.Collections.ForwardListNodeIterator<T, R, P>&&) Member Function

Move constructor.

#### Syntax

```
public ForwardListNodeIterator<T, R, P>(System.Collections.ForwardListNodeIterator<T,
R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.ForwardListNodeIterator<T, R, P>&& | Argument to move from. |

### operator=(System.Collections.ForwardListNodeIterator<T, R, P>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.ForwardListNodeIterator<T, R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.ForwardListNodeIterator<T, R, P>&& | Argument to assign from. |

### ForwardListNodeIterator<T, R, P>(System.Collections.ForwardListNode<T>*) Member Function

Constructor. Constructs a forward list node iterator pointing to a forward list node.

#### Syntax

```
public ForwardListNodeIterator<T, R, P>(System.Collections.ForwardListNode<T>* node_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| node_ | System.Collections.ForwardListNode<T>* | A pointer to a forward list node. |

## Implementation

fwdlist.cm, page 2

### GetNode() const Member Function

Returns the contained pointer to an element.

#### Syntax

```
public System.Collections.ForwardListNode<T>* GetNode() const;
```

#### Returns

System.Collections.ForwardListNode<T>*

Returns the contained pointer to an element.

#### Implementation

fwdlist.cm, page 2

### operator*() const Member Function

Returns a reference to an element.

#### Syntax

```
public R operator*() const;
```

#### Returns

R

Returns a reference to an element.

#### Implementation

fwdlist.cm, page 2

### operator++() Member Function

Advances the iterator pointing to the next element in the forward list.

#### Syntax

```
public System.Collections.ForwardListNodeIterator<T, R, P>& operator++();
```

#### Returns

System.Collections.ForwardListNodeIterator<T, R, P>&

Returns a reference to the iterator.

#### Implementation

fwdlist.cm, page 2

**operator->() const Member Function**

Returns a pointer to an element.

**Syntax**

```
public P operator->() const;
```

**Returns**

P

Returns a pointer to an element.

**Implementation**

fwdlist.cm, page 2

### 3.7.5.3 Nonmember Functions

| Function | Description |
|---|---|
| operator==(System.Collections.-ForwardListNodeIterator<T, R, P>, System.-Collections.ForwardListNodeIterator<T, R, P>) | Compares two forward list node iterators for equality. |

### operator==(System.Collections.ForwardListNodeIterator<T, R, P>, System.Collections.ForwardList R, P>) Function

Compares two forward list node iterators for equality.

#### Syntax

```
public bool operator==(System.Collections.ForwardListNodeIterator<T, R, P> left,
System.Collections.ForwardListNodeIterator<T, R, P> right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | System.Collections.ForwardListNodeIterator<T, R, P> | The first forward list node iterator. |
| right | System.Collections.ForwardListNodeIterator<T, R, P> | The second forward list node iterator. |

#### Returns

bool

Returns true if both iterators point to same forward list node, or both are End() iterators, false otherwise.

#### Implementation

fwdlist.cm, page 6

### 3.7.6 HashMap<K, T, H, C> Class

An associative container of key-value pairs organized in a hash table. The keys need not be ordered.

#### Syntax

```
public class HashMap<K, T, H, C>;
```

#### Constraint

K is Semiregular and T is Semiregular and HashFunction<H, K> and C is Relation and C.Domain is K

#### Model of

ForwardContainer<T>

#### Default Template Arguments

H = System.Collections.Hasher<K>

C = System.EqualTo<K>

#### 3.7.6.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| Compare | C | A relation used to compare keys. |
| ConstIterator | System.Collections.HashtableIterator<-System.Pair<K, T>, const System.Pair<K, T>&, const System.Pair<K, T>*> | A contant iterator type. |
| HashFun | H | A hash function type. |
| Iterator | System.Collections.HashtableIterator<-System.Pair<K, T>, System.Pair<K, T>&, System.Pair<K, T>*> | An iterator type. |
| KeyType | K | The type of the key. |
| MappedType | T | The type associated with the key. |
| ValueType | System.Pair<K, T> | A pair composed of key type and mapped type. |

#### 3.7.6.2 Member Functions

| Member Function | Description |
|---|---|
| HashMap<K, T, H, C>() | Default constructor. |
| HashMap<K, T, H, C>(const System.-Collections.HashMap<K, T, H, C>&) | Copy constructor. |
| operator=(const System.Collections.HashMap<-K, T, H, C>&) | Copy assignment. |
| HashMap<K, T, H, C>(System.Collections.-HashMap<K, T, H, C>&&) | Move constructor. |
| operator=(System.Collections.HashMap<K, T, H, C>&&) | Move assignment. |
| Begin() | Returns an iterator to the beginning of the hash map, or End() if the hash map is empty. |
| Begin() const | Returns a constant iterator to the beginning of the hash map, or CEnd() const if the hash map is empty. |
| CBegin() const | Returns a constant iterator to the beginning of the hash map, or CEnd() const if the hash map is empty. |
| CEnd() const | Returns a constant iterator pointing one past the end of the hash map. |

| | |
|---|---|
| CFind(const K&) const | Searches a key from the hash map and returns a constant iterator pointing to the found element, or CEnd() const if the key is not found in the hash map. |
| Clear() | Makes the hash map empty. |
| Count() const | Returns the number of elements in the hash map. |
| End() | Returns an iterator pointing one past the end of the hash map. |
| End() const | Returns a constant iterator pointing one past the end of the hash map. |
| Find(const K&) | Searches a key from the hash map and returns an iterator pointing to the found element, or End() if the key is not found in the hash map. |
| Find(const K&) const | Searches a key from the hash map and returns a constant iterator pointing to the found element, or CEnd() const if the key is not found in the hash map. |
| Insert(const System.Pair<K, T>&) | Inserts a key-value pair to the hash map if the hash map does not already contain the key. In that case returns a pair consisting a pair of an iterator pointing to the inserted element and **true**. Otherwise does not insert an element, but returns a pair consisting of an iterator pointing to the previously inserted element and **false**. |
| IsEmpty() const | Returns true if the hash map is empty, false otherwise. |
| Remove(System.Collections.HashtableIterator<-System.Pair<K, T>, System.Pair<K, T>&, System.Pair<K, T>*>) | Removes an element pointed by the given iterator from the hash map. |
| Remove(const K&) | Removes an element with the given key from the hash map. |
| operator[](const K&) | Returns a reference to the value associated with the given key. If there are currently no value associated with the given key, a default constructed value is created and inserted to the hash map. |

### HashMap<K, T, H, C>() Member Function

Default constructor.

#### Syntax

```
public HashMap<K, T, H, C>();
```

### HashMap<K, T, H, C>(const System.Collections.HashMap<K, T, H, C>&) Member Function

Copy constructor.

#### Syntax

```
public HashMap<K, T, H, C>(const System.Collections.HashMap<K, T, H, C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.HashMap<K, T, H, C>& | Argument to copy. |

### operator=(const System.Collections.HashMap<K, T, H, C>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.HashMap<K, T, H, C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.HashMap<K, T, H, C>& | Argument to assign. |

### HashMap<K, T, H, C>(System.Collections.HashMap<K, T, H, C>&&) Member Function

Move constructor.

#### Syntax

```
public HashMap<K, T, H, C>(System.Collections.HashMap<K, T, H, C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.HashMap<K, T, H, C>&& | Argument to move from. |

### operator=(System.Collections.HashMap<K, T, H, C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.HashMap<K, T, H, C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.HashMap<K, T, H, C>&& | Argument to assign from. |

### Begin() Member Function

Returns an iterator to the beginning of the hash map, or End() if the hash map is empty.

### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&,
System.Pair<K, T>*> Begin();
```

### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&, System.Pair<-K, T>*>

Returns an iterator to the beginning of the hash map, or End() if the hash map is empty.

### Begin() const Member Function

Returns a constant iterator to the beginning of the hash map, or CEnd() const if the hash map is empty.

### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K,
T>&, const System.Pair<K, T>*> Begin() const;
```

### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K, T>&, const System.Pair<K, T>*>

Returns a constant iterator to the beginning of the hash map, or CEnd() const if the hash map is empty.

### CBegin() const Member Function

Returns a constant iterator to the beginning of the hash map, or CEnd() const if the hash map is empty.

### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K,
T>&, const System.Pair<K, T>*> CBegin() const;
```

### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K, T>&, const System.Pair<K, T>*>

Returns a constant iterator to the beginning of the hash map, or CEnd() const if the hash map is empty.

### CEnd() const Member Function

Returns a constant iterator pointing one past the end of the hash map.

#### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K,
T>&, const System.Pair<K, T>*> CEnd() const;
```

#### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K, T>&, const System.Pair<K, T>*>

Returns a constant iterator pointing one past the end of the hash map.

### CFind(const K&) const Member Function

Searches a key from the hash map and returns a constant iterator pointing to the found element, or CEnd() const if the key is not found in the hash map.

#### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K,
T>&, const System.Pair<K, T>*> CFind(const K& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const K& | A key to search. |

#### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K, T>&, const System.Pair<K, T>*>

Returns a constant iterator pointing to the found element, or CEnd() const if the key is not found in the hash map.

### Clear() Member Function

Makes the hash map empty.

### Syntax

```
public void Clear();
```

### Count() const Member Function

Returns the number of elements in the hash map.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the hash map.

### End() Member Function

Returns an iterator pointing one past the end of the hash map.

#### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&,
System.Pair<K, T>*> End();
```

#### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&, System.Pair<-K, T>*>

Returns an iterator pointing one past the end of the hash map.

### End() const Member Function

Returns a constant iterator pointing one past the end of the hash map.

### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K,
T>&, const System.Pair<K, T>*> End() const;
```

### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K, T>&, const System.Pair<K, T>*>

Returns a constant iterator pointing one past the end of the hash map.

### Find(const K&) Member Function

Searches a key from the hash map and returns an iterator pointing to the found element, or End() if the key is not found in the hash map.

#### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&,
System.Pair<K, T>*> Find(const K& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const K& | A key to seach. |

#### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&, System.Pair<-K, T>*>

Returns an iterator pointing to the found element, or End() if the key is not found in the hash map.

### Find(const K&) const Member Function

Searches a key from the hash map and returns a constant iterator pointing to the found element, or CEnd() const if the key is not found in the hash map.

#### Syntax

```
public System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K,
T>&, const System.Pair<K, T>*> Find(const K& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const K& | A key to search. |

#### Returns

System.Collections.HashtableIterator<System.Pair<K, T>, const System.Pair<K, T>&, const System.Pair<K, T>*>

Returns a constant iterator pointing to the found element, or CEnd() const if the key is not found in the hash map.

### Insert(const System.Pair<K, T>&) Member Function

Inserts a key-value pair to the hash map if the hash map does not already contain the key. In that case returns a pair consisting a pair of an iterator pointing to the inserted element and **true**. Otherwise does not insert an element, but returns a pair consisting of an iterator pointing to the previously inserted element and **false**.

#### Syntax

```
public System.Pair<System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K,
T>&, System.Pair<K, T>*>, bool> Insert(const System.Pair<K, T>& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const System.Pair<K, T>& | A key-value pair to insert. |

#### Returns

System.Pair<System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&, System.Pair<K, T>*>, bool>

Returns a pair consisting an iterator pointing to the key-value pair in the map, and a Boolean value indicating whether the element was inserted in the hash map.

### IsEmpty() const Member Function

Returns true if the hash map is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the hash map is empty, false otherwise.

### Remove(System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&, System.Pair<K, T>*>) Member Function

Removes an element pointed by the given iterator from the hash map.

#### Syntax

```
public void Remove(System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K,
T>&, System.Pair<K, T>*> pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.HashtableIterator<System.Pair<K, T>, System.Pair<K, T>&, System.Pair<K, T>*> | Iterator pointing to an element to remove. |

### Remove(const K&) Member Function

Removes an element with the given key from the hash map.

#### Syntax

```
public void Remove(const K& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const K& | Key of element to remove. |

#### operator[](const K&) Member Function

Returns a reference to the value associated with the given key. If there are currently no value associated with the given key, a default constructed value is created and inserted to the hash map.

#### Syntax

```
public T& operator[](const K& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const K& | A key. |

#### Returns

T&

A value associated with the key.

### 3.7.6.3 Nonmember Functions

| Function | Description |
|---|---|
| operator==(const System.Collections.-HashMap<K, T, H, C>&, const System.-Collections.HashMap<K, T, H, C>&) | Compares two hash maps for equality and returns true if the first hash map contains same number of equal elements as the second hash map, false otherwise. |

### operator==(const System.Collections.HashMap<K, T, H, C>&, const System.Collections.HashMap<K, T, H, C>&) Function

Compares two hash maps for equality and returns true if the first hash map contains same number of equal elements as the second hash map, false otherwise.

#### Syntax

```
public bool operator==(const System.Collections.HashMap<K, T, H, C>& left, const
System.Collections.HashMap<K, T, H, C>& right);
```

## Constraint

K is Semiregular and T is Semiregular and HashFunction<H, K> and C is Relation and C.Domain is K

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.HashMap<K, T, H, C>& | The first hash map. |
| right | const System.Collections.HashMap<K, T, H, C>& | The second hash map. |

#### Returns

bool

Returns true if the first hash map contains same number of equal elements as the second hash map, false otherwise

### 3.7.7 HashSet<T, H, C> Class

A set of unique elements organized in a hash table. The elements need not be ordered.

#### Syntax

```
public class HashSet<T, H, C>;
```

#### Constraint

T is Semiregular and HashFunction<H, T> and C is Relation and C.Domain is T

#### Model of

ForwardContainer<T>

#### Default Template Arguments

H = System.Collections.Hasher<T>

C = System.EqualTo<T>

#### 3.7.7.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| Compare | C | A relation used to compare elements. |
| ConstIterator | System.Collections.HashtableIterator<T, const T&, const T*> | A constant iterator type. |
| HashFun | H | A hash function type. |
| Iterator | System.Collections.HashtableIterator<T, T&, T*> | An iterator type. |
| KeyType | T | The key type is equal to the ValueType for the hash table. |
| ValueType | T | The type of the element in the hash set. |

#### 3.7.7.2 Member Functions

| Member Function | Description |
|---|---|
| HashSet<T, H, C>() | Default constructor. |
| HashSet<T, H, C>(const System.Collections.HashSet<T, H, C>&) | Copy constructor. |
| operator=(const System.Collections.HashSet<T, H, C>&) | Copy assignment. |
| HashSet<T, H, C>(System.Collections.HashSet<T, H, C>&&) | Move constructor. |
| operator=(System.Collections.HashSet<T, H, C>&&) | Move assignment. |
| Begin() | Returns an iterator pointing to the beginning of the hash set, or End() if the hash set is empty. |
| Begin() const | Returns a constant iterator pointing to the beginning of the hash set, or CEnd() const if the hash set is empty. |
| CBegin() const | Returns a constant iterator pointing to the beginning of the hash set, or CEnd() const if the hash set is empty. |
| CEnd() const | Returns a constant iterator pointing one past the end of the hash set. |
| CFind(const T&) const | Searches an element in the hash set and returns a constant iterator pointing to it if found, or CEnd() const otherwise. |

| | |
|---|---|
| Clear() | Makes the hash set empty. |
| Count() const | Returns the number of elements in the hash set. |
| End() | Returns an iterator pointing one past the end of the hash set. |
| End() const | Returns a constant iterator pointing one past the end of the hash set. |
| Find(const T&) | Searches an element in the hash set and returns an iterator pointing to it if found, or End() iterator otherwise. |
| Find(const T&) const | Searches an element in the hash set and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise. |
| Insert(const T&) | Inserts an element into the hash set, if it is not already there. |
| IsEmpty() const | Returns true if the hash set is empty, false otherwise. |
| Remove(System.Collections.HashtableIterator<-T, T&, T*>) | Removes an element pointed by the given iterator from the hash set. |
| Remove(const T&) | Removes an element from the hash set. If element was not found, does nothing. |

### HashSet<T, H, C>() Member Function

Default constructor.

#### Syntax

```
public HashSet<T, H, C>();
```

### HashSet<T, H, C>(const System.Collections.HashSet<T, H, C>&) Member Function

Copy constructor.

#### Syntax

```
public HashSet<T, H, C>(const System.Collections.HashSet<T, H, C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.HashSet<T, H, C>& | Argument to copy. |

### operator=(const System.Collections.HashSet<T, H, C>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.HashSet<T, H, C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.HashSet<T, H, C>& | Argument to assign. |

### HashSet<T, H, C>(System.Collections.HashSet<T, H, C>&&) Member Function

Move constructor.

#### Syntax

```
public HashSet<T, H, C>(System.Collections.HashSet<T, H, C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.HashSet<T, H, C>&& | Argument to move from. |

### operator=(System.Collections.HashSet<T, H, C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.HashSet<T, H, C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.HashSet<T, H, C>&& | Argument to assign from. |

### Begin() Member Function

Returns an iterator pointing to the beginning of the hash set, or End() if the hash set is empty.

### Syntax

```
public System.Collections.HashtableIterator<T, T&, T*> Begin();
```

### Returns

System.Collections.HashtableIterator<T, T&, T*>

Returns an iterator pointing to the beginning of the hash set, or End() if the hash set is empty.

### Begin() const Member Function

Returns a constant iterator pointing to the beginning of the hash set, or CEnd() const if the hash set is empty.

### Syntax

```
public System.Collections.HashtableIterator<T, const T&, const T*> Begin() const;
```

### Returns

System.Collections.HashtableIterator<T, const T&, const T*>

Returns a constant iterator pointing to the beginning of the hash set, or CEnd() const if the hash set is empty.

### CBegin() const Member Function

Returns a constant iterator pointing to the beginning of the hash set, or CEnd() const if the hash set is empty.

### Syntax

```
public System.Collections.HashtableIterator<T, const T&, const T*> CBegin() const;
```

### Returns

System.Collections.HashtableIterator<T, const T&, const T*>

Returns a constant iterator pointing to the beginning of the hash set, or CEnd() const if the hash set is empty.

### CEnd() const Member Function

Returns a constant iterator pointing one past the end of the hash set.

#### Syntax

```
public System.Collections.HashtableIterator<T, const T&, const T*> CEnd() const;
```

#### Returns

System.Collections.HashtableIterator<T, const T&, const T*>

Returns a constant iterator pointing one past the end of the hash set.

### CFind(const T&) const Member Function

Searches an element in the hash set and returns a constant iterator pointing to it if found, or CEnd() const otherwise.

#### Syntax

```
public System.Collections.HashtableIterator<T, const T&, const T*> CFind(const T&
key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const T& | An element to search. |

#### Returns

System.Collections.HashtableIterator<T, const T&, const T*>

Returns a constant iterator pointing to the found element if the search was successful, or CEnd() const iterator otherwise.

### Clear() Member Function

Makes the hash set empty.

### Syntax

```
public void Clear();
```

### Count() const Member Function

Returns the number of elements in the hash set.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the hash set.

### End() Member Function

Returns an iterator pointing one past the end of the hash set.

#### Syntax

```
public System.Collections.HashtableIterator<T, T&, T*> End();
```

#### Returns

System.Collections.HashtableIterator<T, T&, T*>

Returns an iterator pointing one past the end of the hash set.

### End() const Member Function

Returns a constant iterator pointing one past the end of the hash set.

#### Syntax

```
public System.Collections.HashtableIterator<T, const T&, const T*> End() const;
```

#### Returns

System.Collections.HashtableIterator<T, const T&, const T*>

Returns a constant iterator pointing one past the end of the hash set.

### Find(const T&) Member Function

Searches an element in the hash set and returns an iterator pointing to it if found, or End() iterator otherwise.

#### Syntax

```
public System.Collections.HashtableIterator<T, T&, T*> Find(const T& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const T& | An element to seach. |

#### Returns

System.Collections.HashtableIterator<T, T&, T*>

Returns an iterator pointing to the found element if the search was successful, or End() iterator otherwise.

### Find(const T&) const Member Function

Searches an element in the hash set and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise.

#### Syntax

```
public System.Collections.HashtableIterator<T, const T&, const T*> Find(const T&
key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const T& | An element to seach. |

#### Returns

System.Collections.HashtableIterator<T, const T&, const T*>

Returns a constant iterator pointing to the found element if the search was successful, or CEnd() const iterator otherwise.

### Insert(const T&) Member Function

Inserts an element into the hash set, if it is not already there.

#### Syntax

```
public System.Pair<System.Collections.HashtableIterator<T, T&, T*>, bool> Insert(const
T& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const T& | An element to insert. |

#### Returns

System.Pair<System.Collections.HashtableIterator<T, T&, T*>, bool>

Returns a pair consisting of an iterator pointing to the inserted element and **true** if the element was inserted, or a pair consisting an iterator pointing to an existing element and **false** otherwise.

### IsEmpty() const Member Function

Returns true if the hash set is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the hash set is empty, false otherwise.

### Remove(System.Collections.HashtableIterator<T, T&, T*>) Member Function

Removes an element pointed by the given iterator from the hash set.

#### Syntax

```
public void Remove(System.Collections.HashtableIterator<T, T&, T*> pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.HashtableIterator<T, T&, T*> | An iterator pointing to the element to remove. |

#### Remove(const T&) Member Function

Removes an element from the hash set. If element was not found, does nothing.

#### Syntax

```
public void Remove(const T& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const T& | An element to remove. |

### 3.7.7.3 Nonmember Functions

| Function | Description |
|---|---|
| operator==(const System.Collections.HashSet<-T, H, C>&, const System.Collections.HashSet<-T, H, C>&) | Compares two hash sets for equality and returns true if the first hash set contains same number of equal elements as the second hash set, false otherwise. |

### operator==(const System.Collections.HashSet<T, H, C>&, const System.Collections.HashSet<T, H, C>&) Function

Compares two hash sets for equality and returns true if the first hash set contains same number of equal elements as the second hash set, false otherwise.

#### Syntax

```
public bool operator==(const System.Collections.HashSet<T, H, C>& left, const System.Collections
H, C>& right);
```

## Constraint

T is Semiregular and HashFunction<H, T> and C is Relation and C.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.HashSet<T, H, C>& | The first hash set. |
| right | const System.Collections.HashSet<T, H, C>& | The second hash set. |

#### Returns

bool

Returns true if the first hash set contains same number of equal elements as the second hash set, false otherwise.

## 3.7.8 Hasher<T> Class

Default hash function.

### Syntax

```
public class Hasher<T>;
```

### Model of

HashFunction<T, Key>

### Base Class

System.UnaryFun<T, ulong>

#### 3.7.8.1 Member Functions

| Member Function | Description |
|---|---|
| Hasher<T>() | Default constructor. |
| Hasher<T>(const System.Collections.Hasher<-T>&) | Copy constructor. |
| operator=(const System.Collections.Hasher<T>-&) | Copy assignment. |
| Hasher<T>(System.Collections.Hasher<T>&&) | Move constructor. |
| operator=(System.Collections.Hasher<T>&&) | Move assignment. |
| operator()(const T&) | Calls the overloaded *GetHashCode(T)* function to return a hash code for the given key. |

### Hasher<T>() Member Function

Default constructor.

#### Syntax

```
public Hasher<T>();
```

### Hasher<T>(const System.Collections.Hasher<T>&) Member Function

Copy constructor.

#### Syntax

```
public Hasher<T>(const System.Collections.Hasher<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Hasher<T>& | Argument to copy. |

### operator=(const System.Collections.Hasher<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.Hasher<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Hasher<T>& | Argument to assign. |

### Hasher<T>(System.Collections.Hasher<T>&&) Member Function

Move constructor.

#### Syntax

```
public Hasher<T>(System.Collections.Hasher<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Hasher<T>&& | Argument to move from. |

### operator=(System.Collections.Hasher<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.Hasher<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Hasher<T>&& | Argument to assign from. |

### operator()(const T&) Member Function

Calls the overloaded *GetHashCode(T)* function to return a hash code for the given key.

#### Syntax

```
public ulong operator()(const T& x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | const T& | A key. |

#### Returns

ulong

Returns a hash code for the given key.

#### Remarks

By overloading the GetHashCode(T) function for the key type, one can use the default hash function as is.

### 3.7.9 Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare> Class

A hash table of unique elements used to implement HashMap<K, T, H, C> and HashSet<T, H, C>. The keys of elements need not be ordered.

#### Syntax

```
public class Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>;
```

#### Constraint

KeyType is Semiregular and ValueType is Semiregular and KeySelectionFunction<KeyOfValue, KeyType, ValueType> and HashFunction<HashFun, KeyType> and Compare is Relation and Compare.Domain is KeyType

#### Model of

ForwardContainer<T>

#### Default Template Arguments

HashFun = System.Collections.Hasher<KeyType>

Compare = System.EqualTo<KeyType>

#### Base Class

System.Collections.HashtableBase<ValueType>

#### 3.7.9.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| ConstIterator | System.Collections.HashtableIterator<-ValueType, const ValueType&, const ValueType*> | A constant iterator type. |
| Iterator | System.Collections.HashtableIterator<-ValueType, ValueType&, ValueType*> | An iterator type. |

#### 3.7.9.2 Member Functions

| Member Function | Description |
|---|---|
| Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>() | Default constructor. Constructs an empty hash table. |
| Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>(const System.Collections.-Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>&) | Copy constructor. |
| operator=(const System.Collections.Hashtable<-KeyType, ValueType, KeyOfValue, HashFun, Compare>&) | Copy assignment. |
| Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>(System.Collections.-Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>&&) | Move constructor. |
| operator=(System.Collections.Hashtable<-KeyType, ValueType, KeyOfValue, HashFun, Compare>&&) | Move assignment. |
| Begin() | Returns an iterator to the beginning of the hash table, or End() is the hash table is empty. |
| Begin() const | Returns a constant iterator to the beginning of the hash table, or CEnd() const if the hash table is empty. |
| CBegin() const | Returns a constant iterator to the beginning of the hash table, or CEnd() const if the hash table is empty. |
| CEnd() const | Returns a constant iterator one past the end of the hash table. |
| CFind(const KeyType&) const | Searches an element with the given key from the hash table and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise. |

| | |
|---|---|
| Clear() | Makes the hash table empty. |
| Count() const | Returns the number of elements in the hash table. |
| End() | Returns an iterator pointing one past the end of the hash table. |
| End() const | Returns a constant iterator pointing one past the end of the hash table. |
| Find(const KeyType&) | Searches an element with the given key from the hash table and returns an iterator pointing to it if found, or End() iterator otherwise. |
| Find(const KeyType&) const | Searches an element with the given key from the hash table and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise. |
| GetBucket(int) const | Returns a pointer to first bucket with the given index. |
| GetBucketCount() const | Returns the number of buckets in the hash table. |
| GetBucketIndex(const ValueType&) const | Returns index of bucket with the given element. |
| Insert(const ValueType&) | Inserts an element to the hash table and returns a pair consisting of an iterator to the inserted element and **true** if the element was not already in the hash table, or a pair consisting of an iterator pointing to an existing element and **false** otherwise. |
| IsEmpty() const | Returns true if the hash table is empty, false otherwise. |
| Remove(System.Collections.HashtableIterator<-ValueType, ValueType&, ValueType*>) | Removes an element pointed by the given iterator from the hash table. |
| Remove(const KeyType&) | Removes an element with the given key from the hash table. |
| SetMaxLoadFactor(double) | Sets maximum load factor of the hash table. |
| ~Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>() | Destructor. |

### Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>() Member Function

Default constructor. Constructs an empty hash table.

#### Syntax

```
public Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>();
```

### Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>(const System.Collections.Hashta ValueType, KeyOfValue, HashFun, Compare>&) Member Function

Copy constructor.

#### Syntax

```
public Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>(const System.Collections.Hash
ValueType, KeyOfValue, HashFun, Compare>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>& | A hash table to copy from. |

### operator=(const System.Collections.Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.Hashtable<KeyType, ValueType, KeyOfValue,
HashFun, Compare>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>& | A hash table to assign. |

### Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>(System.Collections.Hashtable<K ValueType, KeyOfValue, HashFun, Compare>&&) Member Function

Move constructor.

#### Syntax

```
public Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>(System.Collections.Hashtable<
ValueType, KeyOfValue, HashFun, Compare>&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.Collections.Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>&& | A hash table to move from. |

### operator=(System.Collections.Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.Hashtable<KeyType, ValueType, KeyOfValue,
HashFun, Compare>&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.Collections.Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>&& | A hash table to move from. |

### Begin() Member Function

Returns an iterator to the beginning of the hash table, or End() is the hash table is empty.

### Syntax

```
public System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*> Begin();
```

### Returns

System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*>

Returns an iterator to the beginning of the hash table, or End() is the hash table is empty.

### Begin() const Member Function

Returns a constant iterator to the beginning of the hash table, or CEnd() const if the hash table is empty.

### Syntax

```
public System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>
Begin() const;
```

### Returns

System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>

Returns a constant iterator to the beginning of the hash table, or CEnd() const if the hash table is empty.

### CBegin() const Member Function

Returns a constant iterator to the beginning of the hash table, or CEnd() const if the hash table is empty.

### Syntax

```
public System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>
CBegin() const;
```

### Returns

System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>

Returns a constant iterator to the beginning of the hash table, or CEnd() const if the hash table is empty.

### CEnd() const Member Function

Returns a constant iterator one past the end of the hash table.

#### Syntax

```
public System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>
CEnd() const;
```

#### Returns

System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>

Returns a constant iterator one past the end of the hash table.

### CFind(const KeyType&) const Member Function

Searches an element with the given key from the hash table and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise.

#### Syntax

```
public System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>
CFind(const KeyType& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const KeyType& | A key to search. |

#### Returns

System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>

Returns a constant iterator pointing to the found element, if the search was successful, or CEnd() const iterator otherwise.

### Clear() Member Function

Makes the hash table empty.

### Syntax

```
public void Clear();
```

### Count() const Member Function

Returns the number of elements in the hash table.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the hash table.

### End() Member Function

Returns an iterator pointing one past the end of the hash table.

### Syntax

```
public System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*> End();
```

### Returns

System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*>

Returns an iterator pointing one past the end of the hash table.

### End() const Member Function

Returns a constant iterator pointing one past the end of the hash table.

#### Syntax

```
public System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>
End() const;
```

#### Returns

System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>

Returns a constant iterator pointing one past the end of the hash table.

### Find(const KeyType&) Member Function

Searches an element with the given key from the hash table and returns an iterator pointing to it if found, or End() iterator otherwise.

#### Syntax

```
public System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*> Find(const
KeyType& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const KeyType& | A key to search. |

#### Returns

System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*>

Returns an iterator pointing to the found element, if the search was successful, or End() iterator otherwise.

### Find(const KeyType&) const Member Function

Searches an element with the given key from the hash table and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise.

#### Syntax

```
public System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>
Find(const KeyType& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const KeyType& | A key to search. |

#### Returns

System.Collections.HashtableIterator<ValueType, const ValueType&, const ValueType*>

Returns a constant iterator pointing to the found element, if the search was successful, or CEnd() const iterator otherwise.

### GetBucket(int) const Member Function

Returns a pointer to first bucket with the given index.

#### Syntax

```
public System.Collections.Bucket<ValueType>* GetBucket(int index) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| index | int | Index of the bucket. |

#### Returns

System.Collections.Bucket<ValueType>*

Returns a pointer to first bucket with the given index.

### GetBucketCount() const Member Function

Returns the number of buckets in the hash table.

#### Syntax

```
public int GetBucketCount() const;
```

#### Returns

int

Returns the number of buckets in the hash table.

### GetBucketIndex(const ValueType&) const Member Function

Returns index of bucket with the given element.

#### Syntax

```
public int GetBucketIndex(const ValueType& value) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const ValueType& | An element. |

#### Returns

int

Returns index of bucket with the given element.

### Insert(const ValueType&) Member Function

Inserts an element to the hash table and returns a pair consisting of an iterator to the inserted element and **true** if the element was not already in the hash table, or a pair consisting of an iterator pointing to an existing element and **false** otherwise.

#### Syntax

```
public System.Pair<System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*>,
bool> Insert(const ValueType& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const ValueType& | A value to insert. |

#### Returns

System.Pair<System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*>, bool>-

Returns a pair consisting of an iterator to the inserted element and **true** if the element was not already in the hash table, or a pair consisting of an iterator pointing to an existing element and **false** otherwise.

### IsEmpty() const Member Function

Returns true if the hash table is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the hash table is empty, false otherwise.

### Remove(System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*>) Member Function

Removes an element pointed by the given iterator from the hash table.

#### Syntax

```
public void Remove(System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*>
pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.HashtableIterator<ValueType, ValueType&, ValueType*> | An iterator pointing to the element to remove. |

### Remove(const KeyType&) Member Function

Removes an element with the given key from the hash table.

#### Syntax

```
public void Remove(const KeyType& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const KeyType& | Key of element to remove. |

### SetMaxLoadFactor(double) Member Function

Sets maximum load factor of the hash table.

#### Syntax

```
public void SetMaxLoadFactor(double maxLoadFactor_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| maxLoadFactor_ | double | New maximum load factor. |

#### Remarks

By default maximum load factor is 0.8.

### ~Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>() Member Function

Destructor.

#### Syntax

```
public ~Hashtable<KeyType, ValueType, KeyOfValue, HashFun, Compare>();
```

### 3.7.10 HashtableBase<T> Class

Implementation detail.

#### Syntax

```
public abstract class HashtableBase<T>;
```

#### Constraint

T is Semiregular

#### 3.7.10.1 Type Definitions

| Name | Type | Description |
|---|---|---|

#### 3.7.10.2 Member Functions

| Member Function | Description |
|---|---|

### 3.7.11 HashtableIterator<T, R, P> Class

A hash table iterator type.

#### Syntax

```
public class HashtableIterator<T, R, P>;
```

#### Model of

ForwardIterator<T>

#### 3.7.11.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| PointerType | P | Type of pointer to hash table element. |
| ReferenceType | R | Type of reference to hash table element. |
| ValueType | T | Type of hash table element. |

#### 3.7.11.2 Member Functions

| Member Function | Description |
|---|---|
| HashtableIterator<T, R, P>() | Default constructor. |
| HashtableIterator<T, R, P>(const System.Collections.HashtableIterator<T, R, P>&) | Copy constructor. |
| operator=(const System.Collections.HashtableIterator<T, R, P>&) | Copy assignment. |
| HashtableIterator<T, R, P>(System.Collections.HashtableIterator<T, R, P>&&) | Move constructor. |
| operator=(System.Collections.HashtableIterator<T, R, P>&&) | Move assignment. |
| GetBucket() const | Returns a pointer to the bucket the iterator points to. |
| HashtableIterator<T, R, P>(System.Collections.HashtableBase<T>*, System.Collections.Bucket<T>*) | Constructor. Initalizes the hash table iterator with the given pointer to hash table and pointer to bucket. |
| operator++() | Advances the iterator to point to next element in the hash table. |
| operator->() const | Returns a pointer to the hash table element the iterator points to. |

### HashtableIterator<T, R, P>() Member Function

Default constructor.

#### Syntax

```
public HashtableIterator<T, R, P>();
```

### HashtableIterator<T, R, P>(const System.Collections.HashtableIterator<T, R, P>&) Member Function

Copy constructor.

#### Syntax

```
public HashtableIterator<T, R, P>(const System.Collections.HashtableIterator<T, R,
P>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.HashtableIterator<T, R, P>-& | Argument to copy. |

### operator=(const System.Collections.HashtableIterator<T, R, P>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.HashtableIterator<T, R, P>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.HashtableIterator<T, R, P>-& | Argument to assign. |

### HashtableIterator<T, R, P>(System.Collections.HashtableIterator<T, R, P>&&) Member Function

Move constructor.

#### Syntax

```
public HashtableIterator<T, R, P>(System.Collections.HashtableIterator<T, R, P>&&
that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.HashtableIterator<T, R, P>&& | Argument to move from. |

### operator=(System.Collections.HashtableIterator<T, R, P>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.HashtableIterator<T, R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.HashtableIterator<T, R, P>&& | Argument to assign from. |

### GetBucket() const Member Function

Returns a pointer to the bucket the iterator points to.

#### Syntax

```
public System.Collections.Bucket<T>* GetBucket() const;
```

#### Returns

System.Collections.Bucket<T>*

Returns a pointer to the bucket the iterator points to.

### HashtableIterator<T, R, P>(System.Collections.HashtableBase<T>*, System.Collections.Bucket<T>
### Member Function

Constructor. Initalizes the hash table iterator with the given pointer to hash table and pointer to bucket.

#### Syntax

```
public HashtableIterator<T, R, P>(System.Collections.HashtableBase<T>* table_, System.Collecti
bucket_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| table_ | System.Collections.HashtableBase<T>* | A pointer to hash table. |
| bucket_ | System.Collections.Bucket<T>* | A pointer to bucket. |

### operator++() Member Function

Advances the iterator to point to next element in the hash table.

#### Syntax

```
public System.Collections.HashtableIterator<T, R, P>& operator++();
```

#### Returns

System.Collections.HashtableIterator<T, R, P>&

Returns the iterator.

**operator->() const Member Function**

Returns a pointer to the hash table element the iterator points to.

**Syntax**

```
public P operator->() const;
```

**Returns**

P

Returns a pointer to the hash table element the iterator points to.

### 3.7.11.3 Nonmember Functions

| Function | Description |
|---|---|
| operator==(const System.Collections.-HashtableIterator<T, R, P>&, const System.-Collections.HashtableIterator<T, R, P>&) | Compares two hash table iterators for equality and returns true if they point to the same hash table node if the table is not empty, or both are CEnd() const iterators otherwise. |

### operator==(const System.Collections.HashtableIterator<T, R, P>&, const System.Collections.Hashta R, P>&) Function

Compares two hash table iterators for equality and returns true if they point to the same hash table node if the table is not empty, or both are CEnd() const iterators otherwise.

#### Syntax

```
public bool operator==(const System.Collections.HashtableIterator<T, R, P>& left,
const System.Collections.HashtableIterator<T, R, P>& right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.HashtableIterator<T, R, P>-& | The first iterator. |
| right | const System.Collections.HashtableIterator<T, R, P>-& | The second iterator. |

#### Returns

bool

Returns true if both point to same hash table node, or both are CEnd() const iterators.

### 3.7.12 LinkedList<T> Class

A doubly linked list class.

#### Syntax

```
public class LinkedList<T>;
```

#### Constraint

T is Regular

#### Model of

BidirectionalContainer<T>

BackInsertionSequence<T>

FrontInsertionSequence<T>

InsertionSequence<T>

#### Base Class

System.Collections.LinkedListBase

#### 3.7.12.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| ConstIterator | System.Collections.-LinkedListNodeIterator<T, const T&, const T*> | Type of constant iterator. |
| Iterator | System.Collections.-LinkedListNodeIterator<T, T&, T*> | Type of iterator. |
| ValueType | T | Type of element the linked list contains. |

#### 3.7.12.2 Member Functions

| Member Function | Description |
|---|---|
| LinkedList<T>() | Default constructor. Constructs an empty linked list. |
| LinkedList<T>(const System.Collections.-LinkedList<T>&) | Copy constructor. |
| operator=(const System.Collections.-LinkedList<T>&) | Copy assignment. |
| LinkedList<T>(System.Collections.LinkedList<-T>&&) | Move constructor. |
| operator=(System.Collections.LinkedList<T>&-&) | Move assignment. |
| Add(const T&) | Adds an element to the end of the linked list. |
| Back() const | Returns a constant reference to the last element in the linked list. |
| Begin() | Returns an iterator pointing to the beginning of the linked list, or End() if the linked list is empty. |
| Begin() const | Returns a constant iterator pointing to the beginning of the linked list, or CEnd() const if the linked list is empty. |
| CBegin() const | Returns a constant iterator pointing to the beginning of the linked list, or CEnd() const if the linked list is empty. |
| CEnd() const | Returns a constant iterator pointing one past the end of the linked list. |
| Clear() | Makes the linked list empty. |
| Count() const | Returns the number of elements in the linked list. |

| | |
|---|---|
| End() | Returns an iterator pointing one past the end of the linked list. |
| End() const | Returns a constant iterator pointing one past the end of the linked list. |
| Front() const | Returns a constant reference to the first element in the linked list. |
| Insert(System.Collections.-LinkedListNodeIterator<T, T&, T*>, const T&) | Inserts an element before position pointed by the given iterator to the linked list. |
| InsertFront(const T&) | Inserts an element to the head of the linked list. |
| IsEmpty() const | Returns true if the linked list is empty, false otherwise. |
| Remove(System.Collections.-LinkedListNodeIterator<T, T&, T*>) | Removes an element pointed by the given iterator from the linked list. |
| Remove(const T&) | Removes elements that are equal to the given value from the linked list. |
| RemoveFirst() | Removes the first element from the linked list. |
| RemoveLast() | Removes the last element from the linked list. |
| ~LinkedList<T>() | Destructor. |

### LinkedList<T>() Member Function

Default constructor. Constructs an empty linked list.

#### Syntax

```
public LinkedList<T>();
```

### LinkedList<T>(const System.Collections.LinkedList<T>&) Member Function

Copy constructor.

#### Syntax

```
public LinkedList<T>(const System.Collections.LinkedList<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.LinkedList<T>& | A linked list to copy from. |

### operator=(const System.Collections.LinkedList<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.LinkedList<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.LinkedList<T>& | A linked list to assign from. |

### LinkedList<T>(System.Collections.LinkedList<T>&&) Member Function

Move constructor.

#### Syntax

```
public LinkedList<T>(System.Collections.LinkedList<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.LinkedList<T>&& | A linked list to move from. |

### operator=(System.Collections.LinkedList<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.LinkedList<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.LinkedList<T>&& | A linked list to move from. |

### Add(const T&) Member Function

Adds an element to the end of the linked list.

#### Syntax

```
public void Add(const T& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const T& | An element to add. |

### Back() const Member Function

Returns a constant reference to the last element in the linked list.

### Syntax

```
public const T& Back() const;
```

### Returns

const T&

Returns a constant reference to the last element in the linked list.

### Begin() Member Function

Returns an iterator pointing to the beginning of the linked list, or End() if the linked list is empty.

### Syntax

```
public System.Collections.LinkedListNodeIterator<T, T&, T*> Begin();
```

### Returns

System.Collections.LinkedListNodeIterator<T, T&, T*>

Returns an iterator pointing to the beginning of the linked list, or End() if the linked list is empty.

### Begin() const Member Function

Returns a constant iterator pointing to the beginning of the linked list, or CEnd() const if the linked list is empty.

### Syntax

```
public System.Collections.LinkedListNodeIterator<T, const T&, const T*> Begin() const;
```

### Returns

System.Collections.LinkedListNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to the beginning of the linked list, or CEnd() const if the linked list is empty.

### CBegin() const Member Function

Returns a constant iterator pointing to the beginning of the linked list, or CEnd() const if the linked list is empty.

### Syntax

```
public System.Collections.LinkedListNodeIterator<T, const T&, const T*> CBegin()
const;
```

### Returns

System.Collections.LinkedListNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to the beginning of the linked list, or CEnd() const if the linked list is empty.

### CEnd() const Member Function

Returns a constant iterator pointing one past the end of the linked list.

#### Syntax

```
public System.Collections.LinkedListNodeIterator<T, const T&, const T*> CEnd() const;
```

#### Returns

System.Collections.LinkedListNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing one past the end of the linked list.

### Clear() Member Function

Makes the linked list empty.

### Syntax

```
public void Clear();
```

### Count() const Member Function

Returns the number of elements in the linked list.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the linked list.

### End() Member Function

Returns an iterator pointing one past the end of the linked list.

#### Syntax

```
public System.Collections.LinkedListNodeIterator<T, T&, T*> End();
```

#### Returns

System.Collections.LinkedListNodeIterator<T, T&, T*>

Returns an iterator pointing one past the end of the linked list.

### End() const Member Function

Returns a constant iterator pointing one past the end of the linked list.

#### Syntax

```
public System.Collections.LinkedListNodeIterator<T, const T&, const T*> End() const;
```

#### Returns

System.Collections.LinkedListNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing one past the end of the linked list.

### Front() const Member Function

Returns a constant reference to the first element in the linked list.

#### Syntax

```
public const T& Front() const;
```

#### Returns

const T&

Returns a constant reference to the first element in the linked list.

### Insert(System.Collections.LinkedListNodeIterator<T, T&, T*>, const T&) Member Function

Inserts an element before position pointed by the given iterator to the linked list.

#### Syntax

```
public System.Collections.LinkedListNodeIterator<T, T&, T*> Insert(System.Collections.LinkedList
T&, T*> pos, const T& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.LinkedListNodeIterator<T, T&, T*> | An iterator pointing to a position before to insert. |
| value | const T& | An element to insert. |

#### Returns

System.Collections.LinkedListNodeIterator<T, T&, T*>

Returns an iterator pointing to the inserted element.

### InsertFront(const T&) Member Function

Inserts an element to the head of the linked list.

#### Syntax

```
public System.Collections.LinkedListNodeIterator<T, T&, T*> InsertFront(const T&
value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const T& | An element to insert. |

#### Returns

System.Collections.LinkedListNodeIterator<T, T&, T*>

Returns an iterator pointing to the inserted element.

### IsEmpty() const Member Function

Returns true if the linked list is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the linked list is empty, false otherwise.

### Remove(System.Collections.LinkedListNodeIterator<T, T&, T*>) Member Function

Removes an element pointed by the given iterator from the linked list.

#### Syntax

```
public void Remove(System.Collections.LinkedListNodeIterator<T, T&, T*> pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.LinkedListNodeIterator<T, T&, T*> | An iterator pointing to the element to remove. |

### Remove(const T&) Member Function

Removes elements that are equal to the given value from the linked list.

#### Syntax

```
public void Remove(const T& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const T& | A value to remove. |

### RemoveFirst() Member Function

Removes the first element from the linked list.

#### Syntax

```
public void RemoveFirst();
```

### RemoveLast() Member Function

Removes the last element from the linked list.

### Syntax

```
public void RemoveLast();
```

**~LinkedList<T>() Member Function**

Destructor.

**Syntax**

```
public ~LinkedList<T>();
```

### 3.7.12.3 Nonmember Functions

| Function | | | Description |
|---|---|---|---|
| operator<(const LinkedList<T>&, const LinkedList<T>&) | | System.Collections.-System.Collections.- | Compares two linked listys for less than relationship and returns true if the first linked list comes lexicographically before the second linked list, false otherwise. |
| operator==(const LinkedList<T>&, const LinkedList<T>&) | | System.Collections.-System.Collections.- | Compares two linked lists for equality and returns true if they contain same number of pairwise equal elements, false otherwise. |

### operator<(const System.Collections.LinkedList<T>&, const System.Collections.LinkedList<T>&) Function

Compares two linked listys for less than relationship and returns true if the first linked list comes lexicographically before the second linked list, false otherwise.

#### Syntax

```
public bool operator<(const System.Collections.LinkedList<T>& left, const System.Collections.Li
right);
```

## Constraint

T is TotallyOrdered

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.LinkedList<T>& | The first linked list. |
| right | const System.Collections.LinkedList<T>& | The second linked list. |

#### Returns

bool

Returns true if the first linked list comes lexicographically before the second linked list, false otherwise.

### operator==(const System.Collections.LinkedList<T>&, const System.Collections.LinkedList<T>&) Function

Compares two linked lists for equality and returns true if they contain same number of pairwise equal elements, false otherwise.

#### Syntax

```
public bool operator==(const System.Collections.LinkedList<T>& left, const System.Collections.Li
right);
```

## Constraint

T is Regular

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.LinkedList<T>& | The first linked list. |
| right | const System.Collections.LinkedList<T>& | The second linked list-. |

#### Returns

bool

Returns true if both linked lists contain the same number of pairwise equal elements, false otherwise.

## 3.7.13 LinkedListBase Class

Implementation detail.

### Syntax

```
public abstract class LinkedListBase;
```

#### 3.7.13.1 Member Functions

| Member Function | Description |
|---|---|

### 3.7.14 LinkedListNode<T> Class

A linked list node type.

#### Syntax

```
public class LinkedListNode<T>;
```

#### Base Class

System.Collections.LinkedListNodeBase

#### 3.7.14.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| ValueType | T | Type of linked list element. |

#### 3.7.14.2 Member Functions

| Member Function | Description |
|---|---|
| LinkedListNode<T>() | Default constructor. |
| LinkedListNode<T>(const System.Collections.-LinkedListNode<T>&) | Copy constructor. |
| operator=(const System.Collections.-LinkedListNode<T>&) | Copy assignment. |
| LinkedListNode<T>(System.Collections.-LinkedListNode<T>&&) | Move constructor. |
| operator=(System.Collections.LinkedListNode<-T>&&) | Move assignment. |
| LinkedListNode<T>(const T&, System.-Collections.LinkedListNodeBase*, System.-Collections.LinkedListNodeBase*) | Constructor. Initializes the linked list node with the given value and pointers to previous and next linked list nodes. |
| Value() | Returns a reference to the contained linked list element. |
| Value() const | Returns a constant reference to the contained linked list element. |

### LinkedListNode<T>() Member Function

Default constructor.

#### Syntax

```
public LinkedListNode<T>();
```

### LinkedListNode<T>(const System.Collections.LinkedListNode<T>&) Member Function

Copy constructor.

#### Syntax

```
public LinkedListNode<T>(const System.Collections.LinkedListNode<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.LinkedListNode<T>& | Argument to copy. |

### operator=(const System.Collections.LinkedListNode<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.LinkedListNode<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.LinkedListNode<T>& | Argument to assign. |

### LinkedListNode<T>(System.Collections.LinkedListNode<T>&&) Member Function

Move constructor.

#### Syntax

```
public LinkedListNode<T>(System.Collections.LinkedListNode<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.LinkedListNode<T>&& | Argument to move from. |

### operator=(System.Collections.LinkedListNode<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.LinkedListNode<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.LinkedListNode<T>&& | Argument to assign from. |

### LinkedListNode<T>(const T&, System.Collections.LinkedListNodeBase*, System.Collections.LinkedL
### Member Function

Constructor. Initializes the linked list node with the given value and pointers to previous and next linked list nodes.

#### Syntax

```
public LinkedListNode<T>(const T& value_, System.Collections.LinkedListNodeBase*
prev_, System.Collections.LinkedListNodeBase* next_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value_ | const T& | A linked list element. |
| prev_ | System.Collections.LinkedListNodeBase* | A pointer to previous linked list node. |
| next_ | System.Collections.LinkedListNodeBase* | A pointer to next linked list node. |

### Value() Member Function

Returns a reference to the contained linked list element.

### Syntax

```
public T& Value();
```

### Returns

T&

Returns a reference to the contained linked list element.

### Value() const Member Function

Returns a constant reference to the contained linked list element.

#### Syntax

```
public const T& Value() const;
```

#### Returns

const T&

Returns a constant reference to the contained linked list element.

## 3.7.15 LinkedListNodeBase Class

Implementation detail.

### Syntax

```
public class LinkedListNodeBase;
```

#### 3.7.15.1 Member Functions

| Member Function | Description |
|---|---|
| LinkedListNodeBase() | Default constructor. |

### LinkedListNodeBase() Member Function

Default constructor.

#### Syntax

```
public LinkedListNodeBase();
```

### 3.7.16 LinkedListNodeIterator<T, R, P> Class

A linked list iterator type.

#### Syntax

```
public class LinkedListNodeIterator<T, R, P>;
```

#### Model of

BidirectionalIterator<T>

#### 3.7.16.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| PointerType | P | Type of pointer to linked list element. |
| ReferenceType | R | Type of reference to linked list element. |
| ValueType | T | Type of linked list element. |

#### 3.7.16.2 Member Functions

| Member Function | Description |
|---|---|
| LinkedListNodeIterator<T, R, P>() | Default constructor. |
| LinkedListNodeIterator<T, R, P>(const System.Collections.LinkedListNodeIterator<-T, R, P>&) | Copy constructor. |
| operator=(const System.Collections.-LinkedListNodeIterator<T, R, P>&) | Copy assignment. |
| LinkedListNodeIterator<T, R, P>(System.-Collections.LinkedListNodeIterator<T, R, P>&&) | Move constructor. |
| operator=(System.Collections.-LinkedListNodeIterator<T, R, P>&&) | Move assignment. |
| GetNode() const | Returns a pointer to linked list node the iterator points to. |
| LinkedListNodeIterator<T, R, P>(System.-Collections.LinkedListBase*, System.-Collections.LinkedListNode<T>*) | Constructor. Initializes the linked list node iterator with the given pointer to linked list and pointer to linked list node. |
| operator*() const | Returns a reference to the linked list element the iterator points to. |
| operator++() | Advances the linked list node iterator to point to next element in the linked list. |
| operator–() | Backs up the linked list node iterator to point to previous element in the linked list. |
| operator->() const | Returns a pointer the linked list element the iterator points to. |

### LinkedListNodeIterator<T, R, P>() Member Function

Default constructor.

#### Syntax

```
public LinkedListNodeIterator<T, R, P>();
```

### LinkedListNodeIterator<T, R, P>(const System.Collections.LinkedListNodeIterator<T, R, P>&) Member Function

Copy constructor.

#### Syntax

```
public LinkedListNodeIterator<T, R, P>(const System.Collections.LinkedListNodeIterator<T,
R, P>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.LinkedListNodeIterator<T, R, P>& | Argument to copy. |

### operator=(const System.Collections.LinkedListNodeIterator<T, R, P>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.LinkedListNodeIterator<T, R, P>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.LinkedListNodeIterator<T, R, P>& | Argument to assign. |

### LinkedListNodeIterator<T, R, P>(System.Collections.LinkedListNodeIterator<T, R, P>&&) Member Function

Move constructor.

#### Syntax

```
public LinkedListNodeIterator<T, R, P>(System.Collections.LinkedListNodeIterator<T,
R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.LinkedListNodeIterator<T, R, P>-&& | Argument to move from. |

### operator=(System.Collections.LinkedListNodeIterator<T, R, P>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.LinkedListNodeIterator<T, R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.LinkedListNodeIterator<T, R, P>-&& | Argument to assign from. |

### GetNode() const Member Function

Returns a pointer to linked list node the iterator points to.

#### Syntax

```
public System.Collections.LinkedListNode<T>* GetNode() const;
```

#### Returns

System.Collections.LinkedListNode<T>*

Returns a pointer to linked list node the iterator points to.

### LinkedListNodeIterator<T, R, P>(System.Collections.LinkedListBase*, System.Collections.LinkedLis
### Member Function

Constructor. Initializes the linked list node iterator with the given pointer to linked list and pointer to linked list node.

#### Syntax

```
public LinkedListNodeIterator<T, R, P>(System.Collections.LinkedListBase* list_,
System.Collections.LinkedListNode<T>* node_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| list_ | System.Collections.LinkedListBase* | A pointer to linked list. |
| node_ | System.Collections.LinkedListNode<T>* | A pointer to linked list node. |

### operator*() const Member Function

Returns a reference to the linked list element the iterator points to.

#### Syntax

```
public R operator*() const;
```

#### Returns

R

Returns a reference to the linked list element the iterator points to.

### operator++() Member Function

Advances the linked list node iterator to point to next element in the linked list.

#### Syntax

```
public System.Collections.LinkedListNodeIterator<T, R, P>& operator++();
```

#### Returns

System.Collections.LinkedListNodeIterator<T, R, P>&

Returns the iterator.

### operator–() Member Function

Backs up the linked list node iterator to point to previous element in the linked list.

### Syntax

```
public System.Collections.LinkedListNodeIterator<T, R, P>& operator--();
```

### Returns

System.Collections.LinkedListNodeIterator<T, R, P>&

Returns the iterator.

**operator->() const Member Function**

Returns a pointer the linked list element the iterator points to.

**Syntax**

```
public P operator->() const;
```

**Returns**

P

Returns a pointer the linked list element the iterator points to.

### 3.7.16.3 Nonmember Functions

| Function | Description |
|---|---|
| operator==(const System.Collections.LinkedListNodeIterator<T, R, P>&, const System.Collections.LinkedListNodeIterator<T, R, P>&) | Compares two linked list node iterators and returns true if they point to same linked list node, or both are CEnd() const iterators, false otherwise. |

### operator==(const System.Collections.LinkedListNodeIterator<T, R, P>&, const System.Collections.L R, P>&) Function

Compares two linked list node iterators and returns true if they point to same linked list node, or both are CEnd() const iterators, false otherwise.

#### Syntax

```
public bool operator==(const System.Collections.LinkedListNodeIterator<T, R, P>&
left, const System.Collections.LinkedListNodeIterator<T, R, P>& right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.LinkedListNodeIterator<T, R, P>& | The first linked list node iterator. |
| right | const System.Collections.LinkedListNodeIterator<T, R, P>& | The second linked list node iterator. |

#### Returns

bool

Returns true if both iterators point to same linked list node, or both are CEnd() const iterators, false otherwise.

## 3.7.17 List<T> Class

A container of elements in which the contained elements are in consecutive locations in memory.

### Syntax

```
public class List<T>;
```

### Constraint

T is Semiregular

### Model of

BackInsertionSequence<T>

InsertionSequence<T>

FrontInsertionSequence<T>

RandomAccessContainer<T>

#### 3.7.17.1 Example

```
using System;
using System.Collections;

//  Writes:
//  0, 1, 2
//  1
//  0, 2
//  0, 3

void main()
{
    List<int> intList;
    intList.Add(0);
    intList.Add(2);
    intList.Insert(intList.Begin() + 1, 1);
    Console.Out() << intList << endl();
    int first = intList.RemoveFirst();
    #assert(first == 0);
    int last = intList.RemoveLast();
    #assert(last == 2);
    #assert(intList.Count() == 1);
    Console.Out() << intList << endl();
    intList.InsertFront(0);
    intList.Add(2);
    int one = intList.Remove(intList.Begin() + 1);
    #assert(one == 1);
    Console.Out() << intList << endl();
    int zero = intList[0];
    #assert(zero == 0);
    intList[1] = 3;
    Console.Out() << intList << endl();
    intList.Clear();
    #assert(intList.IsEmpty());
}
```

### 3.7.17.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| ConstIterator | System.RandomAccessIter<T, const T&-, const T*> | A constant iterator type. |
| Iterator | System.RandomAccessIter<T, T&, T*> | An iterator type. |
| ValueType | T | The type of the contained element. |

### 3.7.17.3 Member Functions

| Member Function | Description |
|---|---|
| List<T>() | Constructor. Constructs an empty list. |
| List<T>(const System.Collections.List<T>&) | Copy constructor. |
| operator=(const System.Collections.List<T>&) | Copy assignment. |
| List<T>(System.Collections.List<T>&&) | Move constructor. |
| operator=(System.Collections.List<T>&&) | Move assignment. |
| Add(T&&) | Moves an element to the end of the list. |
| Add(const T&) | Copies an element to the end of the list. |
| Back() | Returns a reference to the last element in the list. |
| Back() const | Returns a constant reference to the last element in the list. |
| Begin() | Returns an iterator pointing to the first element of the list, or End() if the list is empty. |
| Begin() const | Returns a constant iterator pointing to the first element of the list, or CEnd() const if the list is empty. |
| CBegin() const | Returns a constant iterator pointing to the first element of the list, or CEnd() const if the list is empty. |
| CEnd() const | Returns a constant iterator pointing one past the end of the list. |
| Capacity() const | Returns the number of elements that the list can contain without a memory allocation. |

| | |
|---|---|
| Clear() | Makes the list empty. |
| Count() const | Returns the number of elements in the list. |
| End() | Returns an iterator pointing one past the end of the list. |
| End() const | Returns a constant iterator pointing one past the end of the list. |
| Front() | Returns a reference to the first element in the list. |
| Front() const | Returns a constant reference to the first element in the list. |
| Insert(System.RandomAccessIter<T, T&, T*>-, T&&) | Moves an element before the given position in the list. |
| Insert(System.RandomAccessIter<T, T&, T*>-, const T&) | Copies an element before the given position in the list. |
| InsertFront(T&&) | Moves an element to the head of the list. |
| InsertFront(const T&) | Copies an element to the head of the list. |
| IsEmpty() const | Returns true if the list is empty, false otherwise. |
| List<T>(int, const T&) | Constructs a list consisting of given number of copies of given value. |
| Remove(System.RandomAccessIter<T, T&, T*>) | Removes an element pointed by the given iterator from the list and returns the removed element. |
| RemoveFirst() | Removes the first element from the list. |
| RemoveLast() | Removes the last element from the list. |
| Reserve(int) | Makes the capacity of the list at least the given number of elements. |
| Resize(int) | Grows or shrinks the list so that it contains given number of elements. |
| operator[](int) | Returns a reference to the element with the given index. |

| | |
|---|---|
| operator[](int) const | Returns a constant reference to the element with the given index. |
| ~List<T>() | Destructor. |

### List<T>() Member Function

Constructor. Constructs an empty list.

#### Syntax

```
public List<T>();
```

## Implementation

list.cm, page 1

### List<T>(const System.Collections.List<T>&) Member Function

Copy constructor.

#### Syntax

```
public List<T>(const System.Collections.List<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.List<T>& | A list to copy from. |

### operator=(const System.Collections.List<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.List<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.List<T>& | A list to assign from. |

### List<T>(System.Collections.List<T>&&) Member Function

Move constructor.

#### Syntax

```
public List<T>(System.Collections.List<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.List<T>&& | A list to move from. |

### operator=(System.Collections.List<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.List<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.List<T>&& | A list to move from. |

### Add(T&&) Member Function

Moves an element to the end of the list.

#### Syntax

```
public void Add(T&& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| item | T&& | An element to move. |

### Add(const T&) Member Function

Copies an element to the end of the list.

#### Syntax

```
public void Add(const T& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| item | const T& | An element to add. |

### Back() Member Function

Returns a reference to the last element in the list.

### Syntax

```
public T& Back();
```

### Returns

T&

Returns a reference to the last element in the list.

### Implementation

list.cm, page 6

### Back() const Member Function

Returns a constant reference to the last element in the list.

### Syntax

```
public const T& Back() const;
```

### Returns

const T&

Returns a constant reference to the last element in the list.

### Implementation

list.cm, page 6

### Begin() Member Function

Returns an iterator pointing to the first element of the list, or End() if the list is empty.

### Syntax

```
public System.RandomAccessIter<T, T&, T*> Begin();
```

### Returns

System.RandomAccessIter<T, T&, T*>

Returns an iterator pointing to the first element of the list, or End() if the list is empty.

### Implementation

list.cm, page 5

### Begin() const Member Function

Returns a constant iterator pointing to the first element of the list, or CEnd() const if the list is empty.

### Syntax

```
public System.RandomAccessIter<T, const T&, const T*> Begin() const;
```

### Returns

System.RandomAccessIter<T, const T&, const T*>

Returns a constant iterator pointing to the first element of the list, or CEnd() const if the list is empty.

### Implementation

list.cm, page 5

### CBegin() const Member Function

Returns a constant iterator pointing to the first element of the list, or CEnd() const if the list is empty.

### Syntax

```
public System.RandomAccessIter<T, const T&, const T*> CBegin() const;
```

### Returns

System.RandomAccessIter<T, const T&, const T*>

Returns a constant iterator pointing to the first element of the list, or CEnd() const if the list is empty.

### Implementation

list.cm, page 5

### CEnd() const Member Function

Returns a constant iterator pointing one past the end of the list.

#### Syntax

```
public System.RandomAccessIter<T, const T&, const T*> CEnd() const;
```

#### Returns

System.RandomAccessIter<T, const T&, const T*>

Returns a constant iterator pointing one past the end of the list.

#### Implementation

list.cm, page 5

### Capacity() const Member Function

Returns the number of elements that the list can contain without a memory allocation.

### Syntax

```
public int Capacity() const;
```

### Returns

int

Returns the number of elements that the list can contain without a memory allocation.

### Implementation

list.cm, page 2

### Clear() Member Function

Makes the list empty.

### Syntax

```
public void Clear();
```

### Implementation

list.cm, page 3

### Count() const Member Function

Returns the number of elements in the list.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the list.

#### Implementation

list.cm, page 3

### End() Member Function

Returns an iterator pointing one past the end of the list.

### Syntax

```
public System.RandomAccessIter<T, T&, T*> End();
```

### Returns

System.RandomAccessIter<T, T&, T*>

Returns an iterator pointing one past the end of the list.

### Implementation

list.cm, page 5

### End() const Member Function

Returns a constant iterator pointing one past the end of the list.

### Syntax

```
public System.RandomAccessIter<T, const T&, const T*> End() const;
```

### Returns

System.RandomAccessIter<T, const T&, const T*>

Returns a constant iterator pointing one past the end of the list.

### Implementation

list.cm, page 5

### Front() Member Function

Returns a reference to the first element in the list.

### Syntax

```
public T& Front();
```

### Returns

T&

Returns a reference to the first element int the list.

### Implementation

list.cm, page 5

### Front() const Member Function

Returns a constant reference to the first element in the list.

### Syntax

```
public const T& Front() const;
```

### Returns

const T&

Returns a constant reference to the first element in the list.

### Implementation

list.cm, page 5

### Insert(System.RandomAccessIter<T, T&, T*>, T&&) Member Function

Moves an element before the given position in the list.

#### Syntax

```
public System.RandomAccessIter<T, T&, T*> Insert(System.RandomAccessIter<T, T&, T*>
pos, T&& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.RandomAccessIter<T, T&, T*> | An iterator pointing to the position before which to insert. |
| item | T&& | An element to move. |

#### Returns

System.RandomAccessIter<T, T&, T*>

Returns an iterator pointing to the inserted element.

### Insert(System.RandomAccessIter<T, T&, T*>, const T&) Member Function

Copies an element before the given position in the list.

#### Syntax

```
public System.RandomAccessIter<T, T&, T*> Insert(System.RandomAccessIter<T, T&, T*>
pos, const T& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.RandomAccessIter<T, T&, T*> | An iterator pointing to the position before which to insert. |
| item | const T& | An element to insert. |

#### Returns

System.RandomAccessIter<T, T&, T*>

Returns an iterator pointing to the inserted element.

### InsertFront(T&&) Member Function

Moves an element to the head of the list.

#### Syntax

```
public System.RandomAccessIter<T, T&, T*> InsertFront(T&& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| item | T&& | An element to move. |

#### Returns

System.RandomAccessIter<T, T&, T*>

Returns an iterator pointing to the inserted element.

### InsertFront(const T&) Member Function

Copies an element to the head of the list.

#### Syntax

```
public System.RandomAccessIter<T, T&, T*> InsertFront(const T& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| item | const T& | An element to insert. |

#### Returns

System.RandomAccessIter<T, T&, T*>

Returns an iterator pointing to the inserted element.

### IsEmpty() const Member Function

Returns true if the list is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the list is empty, false otherwise.

#### Implementation

list.cm, page 3

### List<T>(int, const T&) Member Function

Constructs a list consisting of given number of copies of given value.

#### Syntax

```
public List<T>(int n, const T& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| n | int | Number of elements to construct. |
| value | const T& | Element to copy. |

### Remove(System.RandomAccessIter<T, T&, T*>) Member Function

Removes an element pointed by the given iterator from the list and returns the removed element.

#### Syntax

```
public T Remove(System.RandomAccessIter<T, T&, T*> pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.RandomAccessIter<T, T&, T*> | An iterator pointing to the element to remove. |

#### Returns

T

Returns the removed element.

#### Implementation

list.cm, page 4

### RemoveFirst() Member Function

Removes the first element from the list.

#### Syntax

```
public T RemoveFirst();
```

#### Returns

T

Returns the removed element.

#### Implementation

list.cm, page 4

### RemoveLast() Member Function

Removes the last element from the list.

### Syntax

```
public T RemoveLast();
```

### Returns

T

Returns the removed element,.

### Implementation

list.cm, page 4

### Reserve(int) Member Function

Makes the capacity of the list at least the given number of elements.

### Syntax

```
public void Reserve(int minRes);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| minRes | int | The minimum number of elements the list can hold without a memory allocation. |

### Implementation

list.cm, page 2

### Resize(int) Member Function

Grows or shrinks the list so that it contains given number of elements.

### Syntax

```
public void Resize(int newCount);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| newCount | int | New number of elements. |

### operator[](int) Member Function

Returns a reference to the element with the given index.

#### Syntax

```
public T& operator[](int index);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| index | int | An index. |

#### Returns

T&

Returns a reference to the element with the given index.

#### Implementation

list.cm, page 5

### operator[](int) const Member Function

Returns a constant reference to the element with the given index.

#### Syntax

```
public const T& operator[](int index) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| index | int | An index. |

#### Returns

const T&

Returns a constant reference to the element with the given index.

#### Implementation

list.cm, page 5

**~List<T>() Member Function**

Destructor.

**Syntax**

```
public ~List<T>();
```

### 3.7.17.4 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.Collections.List<T>&-, const System.Collections.List<T>&) | Compares two lists for less than relationship and returns true if the first list comes lexicographically before the second list, false otherwise. |
| operator==(const System.Collections.List<T>-&, const System.Collections.List<T>&) | Compares two lists for equality and returns true if both lists contain the same number of pairwise equal elements, false otherwise. |

### operator<(const System.Collections.List<T>&, const System.Collections.List<T>&) Function

Compares two lists for less than relationship and returns true if the first list comes lexicographically before the second list, false otherwise.

#### Syntax

```
public bool operator<(const System.Collections.List<T>& left, const System.Collections.List<T>&
right);
```

## Constraint

T is TotallyOrdered

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.List<T>& | The first list. |
| right | const System.Collections.List<T>& | The second list. |

#### Returns

bool

Returns true if the first list comes lexicographically before the second list, false otherwise.

#### Implementation

list.cm, page 7

### operator==(const System.Collections.List<T>&, const System.Collections.List<T>&) Function

Compares two lists for equality and returns true if both lists contain the same number of pairwise equal elements, false otherwise.

#### Syntax

```
public bool operator==(const System.Collections.List<T>& left, const System.Collections.List<T>
right);
```

## Constraint

T is Regular

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.List<T>& | The first list. |
| right | const System.Collections.List<T>& | The second list. |

#### Returns

bool

Returns true if both lists contain the same number of pairwise equal elements, false otherwise.

#### Implementation

list.cm, page 7

## 3.7.18 Map<Key, Value, KeyCompare> Class

An associative container of key-value pairs organized in a red-black tree. The keys need to be ordered.

### Syntax

```
public class Map<Key, Value, KeyCompare>;
```

### Constraint

Key is Semiregular and Value is Semiregular and KeyCompare is Relation and KeyCompare.Domain is Key

### Model of

BidirectionalContainer<T>

### Default Template Arguments

KeyCompare = System.Less<Key>

#### 3.7.18.1 Example

```
using System;
using System.Collections;

//  Writes:
//  the phone number of Stepanov, Alexander is 765432
//  Knuth already inserted
//  Dijkstra, Edsger W. : 111222
//  Knuth, Donald E. : 999888
//  Stepanov, Alexander : 765432
//  Stroustrup, Bjarne : 123456
//  Turing, Alan : 555444

void main()
{
    Map<string, int> phoneBook;
    phoneBook["Stroustrup, Bjarne"] = 123456;
    phoneBook["Stepanov, Alexander"] = 765432;
    phoneBook["Knuth, Donald E."] = 999888;
    phoneBook["Dijkstra, Edsger W."] = 111222;
    phoneBook.Insert(MakePair(string("Turing, Alan"), 555444));

    Map<string, int>.Iterator s = phoneBook.Find("Stepanov, Alexander");
    if (s != phoneBook.End())
    {
        Console.Out() << "the phone number of " << s->first << " is " << s->
            second << endl();
    }
    else
    {
        Console.Error() << "phone number not found" << endl();
    }
```

```
    if (!phoneBook.Insert(MakePair(string("Knuth, Donald E."), 999888)).
        second)
    {
        Console.Out() << "Knuth already inserted" << endl();
    }

    for (const Pair<string, int>& p : phoneBook)
    {
        Console.Out() << p.first << " : " << p.second << endl();
    }
}
```

### 3.7.18.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| Compare | KeyCompare | The type of a relation used to compare keys. |
| ConstIterator | System.Collections.-RedBlackTreeNodeIterator<System.Pair<-Key, Value>, const System.Pair<Key, Value>&, const System.Pair<Key, Value>-*> | A constant iterator type. |
| Iterator | System.Collections.-RedBlackTreeNodeIterator<System.Pair<-Key, Value>, System.Pair<Key, Value>&, System.Pair<Key, Value>*> | An iterator type. |
| KeyType | Key | The type of key. |
| MappedType | Value | The type associated with the key. |
| ValueType | System.Pair<Key, Value> | A pair composed of key type and mapped type. |

### 3.7.18.3 Member Functions

| Member Function | Description |
|---|---|
| Map<Key, Value, KeyCompare>() | Default constructor. Constructs an empty map. |
| Map<Key, Value, KeyCompare>(const System.-Collections.Map<Key, Value, KeyCompare>&) | Copy constructor. |
| operator=(const System.Collections.Map<Key, Value, KeyCompare>&) | Copy assignment. |
| Map<Key, Value, KeyCompare>(System.-Collections.Map<Key, Value, KeyCompare>&&) | Move constructor. |
| operator=(System.Collections.Map<Key, Value, KeyCompare>&&) | Move assignment. |
| Begin() | Returns an iterator pointing to the beginning of the map, or End() if the map is empty. |
| Begin() const | Returns a constant iterator pointing to the beginning of the map. |
| CBegin() const | Returns a constant iterator pointing to the beginning of the map, or CEnd() const if the map is empty. |
| CEnd() const | Returns a constant iterator pointing to one past the end of the map. |

| | |
|---|---|
| CFind(const Key&) const | Searches the given key from the map and returns a constant iterator pointing to the found element, or CEnd() const iterator if the key is not found. |
| Clear() | Makes the map empty. |
| Count() const | Returns the number of key-value pairs in the map. |
| End() | Returns an iterator pointing to one past the end of the map. |
| End() const | Returns a constant iterator pointing one past the end of the map. |
| Find(const Key&) | Searches the given key in the map and returns an iterator pointing to it, if found, or an System.-Collections.Map.End iterator otherwise. |
| Find(const Key&) const | Searches the given key in the map and returns a constant iterator pointing to the found element, or CEnd() const iterator if the key is not found. |
| Insert(const System.Pair<Key, Value>&) | Inserts an element to the map if the key of the element is not already found in the map. |
| IsEmpty() const | Returns true if the map is empty, false otherwise. |
| Remove(System.Collections.-RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.Pair<Key, Value>&, System.-Pair<Key, Value>*>) | Removes an element pointed by the given iterator from the map. |
| Remove(const Key&) | Removes a key-value pair associated with the given key from the map. |
| operator[](const Key&) | Returns a reference to the value associated with the given key. If there are currently no value associated with the given key, a default constructed value is created and inserted in the map. |

### Map<Key, Value, KeyCompare>() Member Function

Default constructor. Constructs an empty map.

#### Syntax

```
public Map<Key, Value, KeyCompare>();
```

## Implementation

map.cm, page 1

### Map<Key, Value, KeyCompare>(const System.Collections.Map<Key, Value, KeyCompare>&) Member Function

Copy constructor.

#### Syntax

```
public Map<Key, Value, KeyCompare>(const System.Collections.Map<Key, Value, KeyCompare>&
that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Map<Key, Value, KeyCompare>& | Argument to copy. |

### operator=(const System.Collections.Map<Key, Value, KeyCompare>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.Map<Key, Value, KeyCompare>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Map<Key, Value, KeyCompare>& | Argument to assign. |

### Map<Key, Value, KeyCompare>(System.Collections.Map<Key, Value, KeyCompare>&&) Member Function

Move constructor.

#### Syntax

```
public Map<Key, Value, KeyCompare>(System.Collections.Map<Key, Value, KeyCompare>&&
that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Map<Key, Value, KeyCompare>&&- & | Argument to move from. |

### operator=(System.Collections.Map<Key, Value, KeyCompare>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.Map<Key, Value, KeyCompare>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Map<Key, Value, KeyCompare>&&- & | Argument to assign from. |

### Begin() Member Function

Returns an iterator pointing to the beginning of the map, or End() if the map is empty.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.Pair<Key,
Value>&, System.Pair<Key, Value>*> Begin();
```

### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.Pair<Key, Value>-
&, System.Pair<Key, Value>*>

Returns an iterator pointing to the beginning of the map, or End() if the map is empty.

### Implementation

map.cm, page 1

### Begin() const Member Function

Returns a constant iterator pointing to the beginning of the map.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const
System.Pair<Key, Value>&, const System.Pair<Key, Value>*> Begin() const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const System.Pair<-Key, Value>&, const System.Pair<Key, Value>*>

Returns a constant iterator pointing to the beginning of the map, or CEnd() const if the map is empty.

### CBegin() const Member Function

Returns a constant iterator pointing to the beginning of the map, or CEnd() const if the map is empty.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const
System.Pair<Key, Value>&, const System.Pair<Key, Value>*> CBegin() const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const System.Pair<-Key, Value>&, const System.Pair<Key, Value>*>

Returns a constant iterator pointing to the beginning of the map, or CEnd() const if the map is empty.

### Implementation

map.cm, page 2

### CEnd() const Member Function

Returns a constant iterator pointing to one past the end of the map.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const
System.Pair<Key, Value>&, const System.Pair<Key, Value>*> CEnd() const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const System.Pair<-Key, Value>&, const System.Pair<Key, Value>*>

Returns a constant iterator pointing to one past the end of the map.

### Implementation

map.cm, page 2

### CFind(const Key&) const Member Function

Searches the given key from the map and returns a constant iterator pointing to the found element, or CEnd() const iterator if the key is not found.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const
System.Pair<Key, Value>&, const System.Pair<Key, Value>*> CFind(const Key& key)
const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const Key& | A key to search. |

#### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const System.Pair<-Key, Value>&, const System.Pair<Key, Value>*>

Returns a constant iterator pointing to the found element, or CEnd() const iterator if the key is not found.

### Clear() Member Function

Makes the map empty.

### Syntax

```
public void Clear();
```

### Implementation

map.cm, page 2

### Count() const Member Function

Returns the number of key-value pairs in the map.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of key-value pairs in the map.

#### Implementation

map.cm, page 2

### End() Member Function

Returns an iterator pointing to one past the end of the map.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.Pair<Key,
Value>&, System.Pair<Key, Value>*> End();
```

### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.Pair<Key, Value>-
&, System.Pair<Key, Value>*>

Returns an iterator pointing to one past the end of the map.

### Implementation

map.cm, page 2

### End() const Member Function

Returns a constant iterator pointing one past the end of the map.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const
System.Pair<Key, Value>&, const System.Pair<Key, Value>*> End() const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const System.Pair<-Key, Value>&, const System.Pair<Key, Value>*>

Returns a constant iterator pointing one past the end of the map.

### Find(const Key&) Member Function

Searches the given key in the map and returns an iterator pointing to it, if found, or an System.Collections.Map.End iterator otherwise.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.Pair<Key,
Value>&, System.Pair<Key, Value>*> Find(const Key& key);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| key | const Key& | A key to search. |

### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.Pair<Key, Value>-&, System.Pair<Key, Value>*>

Returns an iterator pointing to the found key-value pair, if the search was successful, or System.Collections.Map.End iterator otherwise.

### Implementation

map.cm, page 2

### Find(const Key&) const Member Function

Searches the given key in the map and returns a constant iterator pointing to the found element, or CEnd() const iterator if the key is not found.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const
System.Pair<Key, Value>&, const System.Pair<Key, Value>*> Find(const Key& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const Key& | A key to seaerch. |

#### Returns

System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, const System.Pair<-Key, Value>&, const System.Pair<Key, Value>*>

Returns a constant iterator pointing to the found element, or CEnd() const iterator if the key is not found.

### Insert(const System.Pair<Key, Value>&) Member Function

Inserts an element to the map if the key of the element is not already found in the map.

#### Syntax

```
public System.Pair<System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>,
System.Pair<Key, Value>&, System.Pair<Key, Value>*>, bool> Insert(const System.Pair<Key,
Value>& value);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| value | const System.Pair<Key, Value>& | A key-value pair to insert. |

#### Returns

System.Pair<System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.-Pair<Key, Value>&, System.Pair<Key, Value>*>, bool>

If the key of the element is not already in the map returns a pair consisting of an iterator pointing to the inserted element and **true**, or a pair consisting of an iterator pointing to an existing element and **false** otherwise.

### IsEmpty() const Member Function

Returns true if the map is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the map is empty, false otherwise.

#### Implementation

map.cm, page 2

### Remove(System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>, System.Pair<Key, Value>&, System.Pair<Key, Value>*>) Member Function

Removes an element pointed by the given iterator from the map.

#### Syntax

```
public void Remove(System.Collections.RedBlackTreeNodeIterator<System.Pair<Key, Value>,
System.Pair<Key, Value>&, System.Pair<Key, Value>*> pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.RedBlackTreeNodeIterator<-System.Pair<Key, Value>, System.Pair<Key, Value>-&, System.Pair<Key, Value>*> | An iterator pointing to the element to remove. |

### Remove(const Key&) Member Function

Removes a key-value pair associated with the given key from the map.

#### Syntax

```
public bool Remove(const Key& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const Key& | A key. |

#### Returns

bool

Returns true if there was a key-value pair for the given key in the map.

#### Implementation

map.cm, page 2

#### operator[](const Key&) Member Function

Returns a reference to the value associated with the given key. If there are currently no value associated with the given key, a default constructed value is created and inserted in the map.

#### Syntax

```
public Value& operator[](const Key& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const Key& | A key. |

#### Returns

Value&

A value associated with the key.

#### Implementation

map.cm, page 2

### 3.7.18.4 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.Collections.Map<Key, Value, KeyCompare>&, const System.Collections.Map<Key, Value, KeyCompare>&) | Compares two maps for less than relationship and returns true if the first map comes lexicographically before the second map, false otherwise. |
| operator==(const System.Collections.Map<Key, Value, KeyCompare>&, const System.Collections.Map<Key, Value, KeyCompare>&) | Compares two maps for equality and returns true if both maps contain the same number of pairwise equal elements, false otherwise. |

### operator<(const System.Collections.Map<Key, Value, KeyCompare>&, const System.Collections.Map<Key, Value, KeyCompare>&) Function

Compares two maps for less than relationship and returns true if the first map comes lexicographically before the second map, false otherwise.

#### Syntax

```
public bool operator<(const System.Collections.Map<Key, Value, KeyCompare>& left,
const System.Collections.Map<Key, Value, KeyCompare>& right);
```

## Constraint

Key is Semiregular and Value is TotallyOrdered and KeyCompare is Relation and KeyCompare.Domain is Key

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.Map<Key, Value, KeyCompare>& | The first map. |
| right | const System.Collections.Map<Key, Value, KeyCompare>& | The second map. |

#### Returns

bool

Returns true if the first map comes lexicographically before the second map, false otherwise.

#### Example

```
using System;
using System.Collections;

//  Writes:
//  !(m1 < m2) && !(m2 < m1) => m1 == m2
//  m1 < m3
//  m4 < m1

void main()
{
    Map<int, string> m1;
    m1[0] = "foo";
    m1[1] = "bar";
    m1[2] = "baz";
    Map<int, string> m2;
    m2[0] = "foo";
    m2[1] = "bar";
    m2[2] = "baz";
    if (m1 < m2)
    {
        Console.Error() << "bug" << endl();
    }
    else if (m2 < m1)
    {
```

```
        Console.Error() << "bug" << endl();
    }
    else if (m1 != m2)
    {
        Console.Error() << "bug" << endl();
    }
    else
    {
        Console.Out() << "!(m1 < m2) && !(m2 < m1) => m1 == m2" << endl();
    }
    Map<int, string> m3;
    m3[0] = "foo";
    m3[1] = "bar";
    m3[2] = "fluffy";
    if (m1 < m3) //    third element of m1 is less than third element of m3
    {
        Console.Out() << "m1 < m3" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
    Map<int, string> m4;
    m4[0] = "foo";
    m4[1] = "bar";
    if (m1 < m4)
    {
        Console.Error() << "bug" << endl();
    }
    else if (m4 < m1)    // m4[0] == m1[0] && m4[1] == m1[1], but m4 has
        fewer elements
    {
        Console.Out() << "m4 < m1" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

### Implementation

map.cm, page 3

### operator==(const System.Collections.Map<Key, Value, KeyCompare>&, const System.Collections.M Value, KeyCompare>&) Function

Compares two maps for equality and returns true if both maps contain the same number of pairwise equal elements, false otherwise.

#### Syntax

```
public bool operator==(const System.Collections.Map<Key, Value, KeyCompare>& left,
const System.Collections.Map<Key, Value, KeyCompare>& right);
```

## Constraint

Key is Regular and Value is Regular and KeyCompare is Relation and KeyCompare.Domain is Key

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.Map<Key, Value, KeyCompare>& | The first map to compare. |
| right | const System.Collections.Map<Key, Value, KeyCompare>& | The second map to compare. |

#### Returns

bool

Returns true if both maps contain the same number of pairwise equal elements, false otherwise.

#### Example

```
using System;
using System.Collections;

//  Writes:
//  m1 == m2
//  m1 != m3
//  m1 != m4

void main()
{
    Map<int, string> m1;
    m1[0] = "foo";
    m1[1] = "bar";
    m1[2] = "baz";
    Map<int, string> m2;
    m2[0] = "foo";
    m2[1] = "bar";
    m2[2] = "baz";
    if (m1 == m2)    // same number of pairwise equal elements
    {
        Console.Out() << "m1 == m2" << endl();
    }
    else
    {
```

```
        Console.Error() << "bug" << endl();
    }
    Map<int, string> m3;
    m3[0] = "foo";
    m3[1] = "bar";
    m3[2] = "fluffy";
    if (m1 != m3) //    third element differ
    {
        Console.Out() << "m1 != m3" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
    Map<int, string> m4;
    m4[0] = "foo";
    m4[1] = "bar";
    if (m1 != m4)   // different number of elements
    {
        Console.Out() << "m1 != m4" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

### Implementation

map.cm, page 3

## 3.7.19 Queue<T> Class

A first-in first-out data structure.

### Syntax

```
public class Queue<T>;
```

### Constraint

T is Semiregular

#### 3.7.19.1 Example

```
using System;
using System.Collections;
using Simulation;

//  Writes:
//  clock: 2: customer 1 arrives
//  clock: 7: customer 1 leaves
//  clock: 15: customer 2 arrives
//  clock: 20: customer 2 leaves
//  clock: 25: customer 3 arrives
//  clock: 30: customer 3 leaves
//  clock: 31: customer 4 arrives
//  clock: 36: customer 4 leaves
//  clock: 46: customer 5 arrives
//  clock: 51: customer 5 leaves
//  clock: 56: customer 6 arrives
//  clock: 61: customer 6 leaves
//  clock: 70: customer 7 arrives
//  clock: 75: customer 7 leaves
//  clock: 84: customer 8 arrives
//  clock: 89: customer 8 leaves
//  clock: 92: customer 9 arrives
//  clock: 97: customer 9 leaves
//  clock: 102: customer 10 arrives
//  clock: 107: customer 10 leaves
//  clock: 107: end of simulation.

namespace Simulation
{
    public class CustomerEvent
    {
        public CustomerEvent(): elapsed(0), customerNumber(0)
        {
        }
        public CustomerEvent(int elapsed_, int customerNumber_): elapsed(
            elapsed_), customerNumber(customerNumber_)
        {
        }
        public int Elapsed() const
        {
            return elapsed;
        }
        public int CustomerNumber() const
```

```
        {
            return customerNumber;
        }
        private int elapsed;
        private int customerNumber;
    }
    public typedef Queue<CustomerEvent> CustomerEventQueue;
}

public const int serviceTime = 5;

void main()
{
    CustomerEventQueue queue;
    int customerNumber = 1;
    int n = 10;
    for (int i = 0; i < n; ++i)
    {
        queue.Put(CustomerEvent(rand() % 10 + 1, customerNumber++));
    }
    int clock = 0;
    while (!queue.IsEmpty())
    {
        CustomerEvent event = queue.Get();
        clock = clock + event.Elapsed();
        Console.Out() << "clock: " << clock << ": customer " << event.
            CustomerNumber() << " arrives" << endl();
        clock  = clock + serviceTime;
        Console.Out() << "clock: " << clock << ": customer " << event.
            CustomerNumber() << " leaves" << endl();
    }
    Console.Out() << "clock: " << clock << ": end of simulation." << endl();
}
```

### 3.7.19.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| ValueType | T | The type of the queue element. |

### 3.7.19.3 Member Functions

| Member Function | Description |
|---|---|
| Queue<T>() | Default constructor. Constructs an empty queue. |
| Queue<T>(const System.Collections.Queue<-T>&) | Copy constructor. |
| operator=(const System.Collections.Queue<T>-&) | Copy assignment. |
| Queue<T>(System.Collections.Queue<T>&&) | Move constructor. |
| operator=(System.Collections.Queue<T>&&) | Move assignment. |
| Clear() | Makes the queue empty. |
| Count() const | Returns the number of elements in the queue. |
| Front() const | Returns a constant reference to the first element in the queue. |
| Get() | Removes the first element from the queue and returns it. |
| IsEmpty() const | Returns true if the queue is empty, false otherwise. |
| Put(T&&) | Moves an element to the back of the queue. |
| Put(const T&) | Puts an element to the back of the queue. |

### Queue<T>() Member Function

Default constructor. Constructs an empty queue.

#### Syntax

```
public Queue<T>();
```

## Implementation

queue.cm, page 1

### Queue<T>(const System.Collections.Queue<T>&) Member Function

Copy constructor.

#### Syntax

```
public Queue<T>(const System.Collections.Queue<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Queue<T>& | Argument to copy. |

### operator=(const System.Collections.Queue<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.Queue<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Queue<T>& | Argument to assign. |

### Queue<T>(System.Collections.Queue<T>&&) Member Function

Move constructor.

#### Syntax

```
public Queue<T>(System.Collections.Queue<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Queue<T>&& | Argument to move from. |

## Implementation

queue.cm, page 1

### operator=(System.Collections.Queue<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.Queue<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Queue<T>&& | Argument to assign from. |

## Implementation

queue.cm, page 1

### Clear() Member Function

Makes the queue empty.

### Syntax

```
public void Clear();
```

## Implementation

queue.cm, page 2

### Count() const Member Function

Returns the number of elements in the queue.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the queue.

#### Implementation

queue.cm, page 1

### Front() const Member Function

Returns a constant reference to the first element in the queue.

### Syntax

```
public const T& Front() const;
```

### Returns

const T&

Returns a constant reference to the first element in the queue.

### Implementation

queue.cm, page 2

### Get() Member Function

Removes the first element from the queue and returns it.

#### Syntax

```
public T Get();
```

#### Returns

T

Returns the removed first element of the queue.

#### Implementation

queue.cm, page 1

### IsEmpty() const Member Function

Returns true if the queue is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the queue is empty, false otherwise.

#### Implementation

queue.cm, page 1

### Put(T&&) Member Function

Moves an element to the back of the queue.

#### Syntax

```
public void Put(T&& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| item | T&& | An element to insert. |

#### Implementation

queue.cm, page 1

### Put(const T&) Member Function

Puts an element to the back of the queue.

#### Syntax

```
public void Put(const T& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| item | const T& | An element to put. |

#### Implementation

queue.cm, page 1

### 3.7.20 RedBlackTree<KeyType, ValueType, KeyOfValue, Compare> Class

A self-balancing binary search tree of unique elements used to implement Set<T, C> and Map<Key, Value, KeyCompare>. The keys of the elements in the tree need to be ordered.

#### Syntax

```
public class RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>;
```

#### Constraint

KeyType is Semiregular and ValueType is Semiregular and KeySelectionFunction<KeyOfValue, KeyType, ValueType> and Compare is Relation and Compare.Domain is KeyType

### 3.7.20.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| ConstIterator | System.Collections.-RedBlackTreeNodeIterator<ValueType, const ValueType&, const ValueType*> | A constant iterator type. |
| Iterator | System.Collections.-RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*> | An iterator type. |

### 3.7.20.2 Member Functions

| Member Function | Description |
|---|---|
| RedBlackTree<KeyType, ValueType, KeyOf-Value, Compare>() | Default constructor. Constructs an empty red-black tree. |
| RedBlackTree<KeyType, ValueType, KeyOf-Value, Compare>(const System.Collections.-RedBlackTree<KeyType, ValueType, KeyOf-Value, Compare>&) | Copy constructor. |
| operator=(const System.Collections.-RedBlackTree<KeyType, ValueType, Key-OfValue, Compare>&) | Copy assignment. |
| RedBlackTree<KeyType, ValueType, Key-OfValue, Compare>(System.Collections.-RedBlackTree<KeyType, ValueType, Key-OfValue, Compare>&&) | Move constructor. |
| operator=(System.Collections.RedBlackTree<-KeyType, ValueType, KeyOfValue, Compare>-&&) | Move assignment. |
| Begin() | Returns a bidirectional iterator pointing to the beginning of the red-black tree. |
| Begin() const | Returns a constant bidirectional iterator pointing to the beginning of the red-black tree. |
| CBegin() const | Returns a constant bidirectional iterator pointing to the beginning of the red-black tree. |
| CEnd() const | Returns a constant bidirectional iterator pointing to one past the end of the red-black tree. |
| CFind(const KeyType&) const | Finds an element with the given key in the red-black tree and returns a constant iterator pointing to it if found, or System.Collections.-RedBlackTree.CEnd iterator otherwise. |

| | |
|---|---|
| Clear() | Makes the red-black tree empty. |
| Count() const | Returns the number of elements in the red-black tree. |
| End() | Returns a bidirectional iterator pointing to one past the end of the red-black tree. |
| End() const | Returns a constant bidirectional iterator pointing to one past the end of the red-black tree. |
| Find(const KeyType&) | Finds an element with the given key in the red-black tree and returns an iterator pointing to it if found, or System.Collections.RedBlackTree.End iterator otherwise. |
| Find(const KeyType&) const | Finds an element with the given key in the red-black tree and returns a constant iterator pointing to it if found, or System.Collections.-RedBlackTree.CEnd.const iterator otherwise. |
| IsEmpty() const | Returns true if the red-black tree is empty, false otherwise. |
| Remove(System.Collections.-RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*>) | Removes an element pointed by the given iterator from the red-black tree. |
| Remove(const KeyType&) | Removes an element with the given key from the red-black tree. If an element with the given is not found, does nothing. |
| ~RedBlackTree<KeyType, ValueType, KeyOf-Value, Compare>() | Destructor. |

### RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>() Member Function

Default constructor. Constructs an empty red-black tree.

#### Syntax

```
public RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>();
```

## Implementation

rbtree.cm, page 12

### RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>(const System.Collections.RedBlackTre ValueType, KeyOfValue, Compare>&) Member Function

Copy constructor.

#### Syntax

```
public RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>(const System.Collections.RedBlackTr
ValueType, KeyOfValue, Compare>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>& | A red-black tree to copy. |

### operator=(const System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue,
Compare>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>& | A red-black tree to assign. |

### RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>(System.Collections.RedBlackTree<Key ValueType, KeyOfValue, Compare>&&) Member Function

Move constructor.

#### Syntax

```
public RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>(System.Collections.RedBlackTree<Ke
ValueType, KeyOfValue, Compare>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>&& | A red-black tree to move from. |

### operator=(System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue,
Compare>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>&& | A red-black tree to assign. |

### Begin() Member Function

Returns a bidirectional iterator pointing to the beginning of the red-black tree.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*>
Begin();
```

### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*>

Returns a bidirectional iterator pointing to the beginning of the red-black tree.

### Implementation

rbtree.cm, page 12

### Begin() const Member Function

Returns a constant bidirectional iterator pointing to the beginning of the red-black tree.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const
ValueType*> Begin() const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const ValueType*>-

Returns a constant bidirectional iterator pointing to the beginning of the red-black tree.

### Implementation

rbtree.cm, page 12

### CBegin() const Member Function

Returns a constant bidirectional iterator pointing to the beginning of the red-black tree.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const
ValueType*> CBegin() const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const ValueType*>-

Returns a constant bidirectional iterator pointing to the beginning of the red-black tree.

### Implementation

rbtree.cm, page 12

### CEnd() const Member Function

Returns a constant bidirectional iterator pointing to one past the end of the red-black tree.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const
ValueType*> CEnd() const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const ValueType*>-

Returns a constant bidirectional iterator pointing to one past the end of the red-black tree.

### Implementation

rbtree.cm, page 13

### CFind(const KeyType&) const Member Function

Finds an element with the given key in the red-black tree and returns a constant iterator pointing to it if found, or System.Collections.RedBlackTree.CEnd iterator otherwise.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const
ValueType*> CFind(const KeyType& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const KeyType& | A key to search. |

#### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const ValueType*>-

Returns a constant iterator pointing to it if found, or System.Collections.RedBlackTree.CEnd iterator otherwise.

### Clear() Member Function

Makes the red-black tree empty.

### Syntax

```
public void Clear();
```

### Implementation

rbtree.cm, page 13

### Count() const Member Function

Returns the number of elements in the red-black tree.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the red-black tree.

#### Implementation

rbtree.cm, page 13

### End() Member Function

Returns a bidirectional iterator pointing to one past the end of the red-black tree.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*>
End();
```

### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*>

Returns a bidirectional iterator pointing to one past the end of the red-black tree.

### Implementation

rbtree.cm, page 13

### End() const Member Function

Returns a constant bidirectional iterator pointing to one past the end of the red-black tree.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const
ValueType*> End() const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const ValueType*>-

Returns a constant bidirectional iterator pointing to one past the end of the red-black tree.

### Implementation

rbtree.cm, page 12

### Find(const KeyType&) Member Function

Finds an element with the given key in the red-black tree and returns an iterator pointing to it if found, or System.Collections.RedBlackTree.End iterator otherwise.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*>
Find(const KeyType& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const KeyType& | A key to search. |

#### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*>

Returns an iterator pointing to the found element if search is successful, or System.Collections.RedBlackTree.End iterator otherwise.

#### Implementation

rbtree.cm, page 13

### Find(const KeyType&) const Member Function

Finds an element with the given key in the red-black tree and returns a constant iterator pointing to it if found, or System.Collections.RedBlackTree.CEnd.const iterator otherwise.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const
ValueType*> Find(const KeyType& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const KeyType& | A key to search. |

#### Returns

System.Collections.RedBlackTreeNodeIterator<ValueType, const ValueType&, const ValueType*>-

Returns a constant iterator pointing to the found element if search is successful, or System.Collections.RedBlackTree.CEnd.const iterator otherwise.

#### Implementation

rbtree.cm, page 14

### IsEmpty() const Member Function

Returns true if the red-black tree is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the red-black tree is empty, false otherwise.

#### Implementation

rbtree.cm, page 13

### Remove(System.Collections.RedBlackTreeNodeIterator<ValueType, ValueType&, ValueType*>) Member Function

Removes an element pointed by the given iterator from the red-black tree.

#### Syntax

```
public void Remove(System.Collections.RedBlackTreeNodeIterator<ValueType, ValueType&,
ValueType*> pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.RedBlackTreeNodeIterator<-ValueType, ValueType&, ValueType*> | An iterator pointing to the element to be removed. |

#### Implementation

rbtree.cm, page 17

### Remove(const KeyType&) Member Function

Removes an element with the given key from the red-black tree. If an element with the given is not found, does nothing.

#### Syntax

```
public bool Remove(const KeyType& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const KeyType& | A key of an element to remove. |

#### Returns

bool

Returns true if an element was removed, false otherwise.

#### Implementation

rbtree.cm, page 16

### ~RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>() Member Function

Destructor.

#### Syntax

```
public ~RedBlackTree<KeyType, ValueType, KeyOfValue, Compare>();
```

#### Implementation

rbtree.cm, page 12

### 3.7.21 RedBlackTreeNodeIterator<T, R, P> Class

A bidirectional iterator that iterates through the elements in a reb-black tree.

#### Syntax

```
public class RedBlackTreeNodeIterator<T, R, P>;
```

#### Model of

BidirectionalIterator<T>

#### 3.7.21.1 Type Definitions

| Name | Type | Description |
|---|---|---|
| PointerType | P | The type of a pointer to an element. |
| ReferenceType | R | The type of a reference to an element. |
| ValueType | T | The type of the element. |

#### 3.7.21.2 Member Functions

| Member Function | Description |
|---|---|
| RedBlackTreeNodeIterator<T, R, P>() | Constructor. Default constructs a red-black tree node iterator. |
| RedBlackTreeNodeIterator<T, R, P>(const System.Collections.RedBlackTreeNodeIterator<-T, R, P>&) | Copy constructor. |
| operator=(const System.Collections.-RedBlackTreeNodeIterator<T, R, P>&) | Copy assignment. |
| RedBlackTreeNodeIterator<T, R, P>(System.-Collections.RedBlackTreeNodeIterator<T, R, P>&&) | Move constructor. |
| operator=(System.Collections.-RedBlackTreeNodeIterator<T, R, P>&&) | Move assignment. |
| GetNode() const | Returns a pointer to a red-black tree node. |
| RedBlackTreeNodeIterator<T, R, P>(System.-Collections.RedBlackTreeNode<T>*) | Constructor. Constructs an iterator pointing to a red-black tree node. |
| operator*() const | Returns a reference to an element. |
| operator++() | Advances the iterator pointing to the next element in the red-black tree. |
| operator–() | Backs the iterator pointing to the previous element in the red-black tree. |
| operator->() const | Returns a pointer to an element. |

### RedBlackTreeNodeIterator<T, R, P>() Member Function

Constructor. Default constructs a red-black tree node iterator.

#### Syntax

```
public RedBlackTreeNodeIterator<T, R, P>();
```

## Implementation

rbtree.cm, page 11

### RedBlackTreeNodeIterator<T, R, P>(const System.Collections.RedBlackTreeNodeIterator<T, R, P>&) Member Function

Copy constructor.

#### Syntax

```
public RedBlackTreeNodeIterator<T, R, P>(const System.Collections.RedBlackTreeNodeIterator<T,
R, P>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.RedBlackTreeNodeIterator<-T, R, P>& | Argument to copy. |

### operator=(const System.Collections.RedBlackTreeNodeIterator<T, R, P>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.RedBlackTreeNodeIterator<T, R, P>&
that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.RedBlackTreeNodeIterator<-T, R, P>& | Argument to assign. |

### RedBlackTreeNodeIterator<T, R, P>(System.Collections.RedBlackTreeNodeIterator<T, R, P>&&) Member Function

Move constructor.

#### Syntax

```
public RedBlackTreeNodeIterator<T, R, P>(System.Collections.RedBlackTreeNodeIterator<T,
R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.RedBlackTreeNodeIterator<T, R, P>&& | Argument to move from. |

### operator=(System.Collections.RedBlackTreeNodeIterator<T, R, P>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.RedBlackTreeNodeIterator<T, R, P>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.RedBlackTreeNodeIterator<T, R, P>&& | Argument to assign from. |

### GetNode() const Member Function

Returns a pointer to a red-black tree node.

#### Syntax

```
public System.Collections.RedBlackTreeNode<T>* GetNode() const;
```

#### Returns

System.Collections.RedBlackTreeNode<T>*

Returns a pointer to a red-black tree node.

#### Implementation

rbtree.cm, page 11

### RedBlackTreeNodeIterator<T, R, P>(System.Collections.RedBlackTreeNode<T>*) Member Function

Constructor. Constructs an iterator pointing to a red-black tree node.

#### Syntax

```
public RedBlackTreeNodeIterator<T, R, P>(System.Collections.RedBlackTreeNode<T>*
node_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| node_ | System.Collections.RedBlackTreeNode<T>* | A pointer to a red-black tree node. |

#### Implementation

rbtree.cm, page 11

### operator*() const Member Function

Returns a reference to an element.

#### Syntax

```
public R operator*() const;
```

#### Returns

R

Returns a reference to an element.

#### Implementation

rbtree.cm, page 11

### operator++() Member Function

Advances the iterator pointing to the next element in the red-black tree.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, R, P>& operator++();
```

### Returns

System.Collections.RedBlackTreeNodeIterator<T, R, P>&

Returns a reference to the iterator.

### Implementation

rbtree.cm, page 11

### operator–() Member Function

Backs the iterator pointing to the previous element in the red-black tree.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, R, P>& operator--();
```

### Returns

System.Collections.RedBlackTreeNodeIterator<T, R, P>&

Returns a reference to the iterator.

### Implementation

rbtree.cm, page 11

**operator->() const Member Function**

Returns a pointer to an element.

**Syntax**

```
public P operator->() const;
```

**Returns**

P

Returns a pointer to an element.

**Implementation**

rbtree.cm, page 11

### 3.7.21.3 Nonmember Functions

| Function | Description |
|---|---|
| operator==(const System.Collections.-RedBlackTreeNodeIterator<T, R, P>&, const System.Collections.RedBlackTreeNodeIterator<-T, R, P>&) | Compares two red-black tree node iterators for equality. |

### operator==(const System.Collections.RedBlackTreeNodeIterator<T, R, P>&, const System.Collections.RedBlackTreeNodeIterator<T, R, P>&) Function

Compares two red-black tree node iterators for equality.

#### Syntax

```
public bool operator==(const System.Collections.RedBlackTreeNodeIterator<T, R, P>&
left, const System.Collections.RedBlackTreeNodeIterator<T, R, P>& right);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.RedBlackTreeNodeIterator<-T, R, P>& | The first red-black tree node iterator. |
| right | const System.Collections.RedBlackTreeNodeIterator<-T, R, P>& | The second red-black tree node iterator. |

#### Returns

bool

Returns true if both iterators point to same red-black tree node, or both are End() iterators, false otherwise.

#### Implementation

rbtree.cm, page 11

## 3.7.22 Set<T, C> Class

A container that contains a set of unique elements organized in a red-black tree. The elements need to be ordered.

### Syntax

```
public class Set<T, C>;
```

### Constraint

T is Semiregular and C is Relation and C.Domain is T

### Model of

BidirectionalContainer<T>

### Default Template Arguments

C = System.Less<T>

#### 3.7.22.1 Example

```
using System;
using System.Collections;

//  Writes:
//  10, 43, 112
//  112 already exists
//  i1 points to item number 0
//  15 not found
//  43 removed
//  112

void main()
{
    Set<int> set;
    set.Insert(43);
    set.Insert(10);
    set.Insert(112);
    Console.Out() << set << endl();
    if (!set.Insert(112).second)
    {
        Console.Out() << 112 << " already exists" << endl();
    }
    Set<int>.Iterator i1 = set.Find(10);
    if (i1 != set.End())
    {
        Console.Out() << "i1 points to item number " << Distance(set.Begin()
            , i1) << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
    Set<int>.Iterator i2 = set.Find(15);
```

```
    if (i2 == set.End())
    {
        Console.Out() << 15 << " not found" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
    bool removed = set.Remove(43);
    if (removed)
    {
        Console.Out() << 43 << " removed" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
    set.Remove(i1);
    Console.Out() << set << endl();
}
```

### 3.7.22.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| Compare | C | A relation used to order elements in the set. |
| ConstIterator | System.Collections.-RedBlackTreeNodeIterator<T, const T&, const T*> | A constant iterator type. |
| Iterator | System.Collections.-RedBlackTreeNodeIterator<T, T&, T*> | An iterator type. |
| KeyType | T | The key type is equal to the System.Collections.Set.-ValueType for the red-black tree. |
| ValueType | T | The type of the element in the set. |

### 3.7.22.3 Member Functions

| Member Function | Description |
|---|---|
| Set<T, C>() | Default constructor. Constructs an empty set. |
| Set<T, C>(const System.Collections.Set<T, C>-&) | Copy constructor. |
| operator=(const System.Collections.Set<T, C>-&) | Copy assignment. |
| Set<T, C>(System.Collections.Set<T, C>&&) | Move constructor. |
| operator=(System.Collections.Set<T, C>&&) | Move assignment. |
| Begin() | Returns an iterator pointing to the beginning of the set, or End() if the set is empty. |
| Begin() const | Returns a constant iterator pointing to the beginnig of the set, or CEnd() const ir the set is empty. |
| CBegin() const | Returns a constant iterator pointing to the beginning of the set, or CEnd() const if the set is empty. |
| CEnd() const | Returns a constant iterator pointing one past the end of the set. |
| CFind(const T&) const | Searches an element from the set and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise. |

| | |
|---|---|
| Clear() | Makes the set empty. |
| Count() const | Returns the number of elements in the set. |
| End() | Returns an iterator pointing one past the end of the set. |
| End() const | Returns a constant iterator pointing one past the end of the set. |
| Find(const T&) | Searches an element from the set and returns an iterator pointing to it if found, or End() iterator otherwise. |
| Find(const T&) const | Searches an element from the set and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise. |
| Insert(const T&) | Inserts an element to the set if it is not already there and returns a pair consisting of an iterator pointing to the inserted element and **true** if element was inserted, and a pair consisting of an iterator pointing to the existing element and **false** otherwise. |
| IsEmpty() const | Returns true if the set is empty, false otherwise. |
| Remove(System.Collections.-RedBlackTreeNodeIterator<T, T&, T*>) | Removes an element pointed by the given iterator from the set. |
| Remove(const T&) | Removes an element from the set. If the element was not found, does nothing. |

### Set<T, C>() Member Function

Default constructor. Constructs an empty set.

#### Syntax

```
public Set<T, C>();
```

## Implementation

set.cm, page 1

### Set<T, C>(const System.Collections.Set<T, C>&) Member Function

Copy constructor.

#### Syntax

```
public Set<T, C>(const System.Collections.Set<T, C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Set<T, C>& | Argument to copy. |

### operator=(const System.Collections.Set<T, C>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.Set<T, C>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Set<T, C>& | Argument to assign. |

### Set<T, C>(System.Collections.Set<T, C>&&) Member Function

Move constructor.

#### Syntax

```
public Set<T, C>(System.Collections.Set<T, C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Set<T, C>&& | Argument to move from. |

### operator=(System.Collections.Set<T, C>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.Set<T, C>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Set<T, C>&& | Argument to assign from. |

### Begin() Member Function

Returns an iterator pointing to the beginning of the set, or End() if the set is empty.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, T&, T*> Begin();
```

### Returns

System.Collections.RedBlackTreeNodeIterator<T, T&, T*>

Returns an iterator pointing to the beginning of the set, or End() if the set is empty.

### Implementation

set.cm, page 1

### Begin() const Member Function

Returns a constant iterator pointing to the beginnig of the set, or CEnd() const ir the set is empty.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*> Begin()
const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to the beginnig of the set, or CEnd() const ir the set is empty.

### CBegin() const Member Function

Returns a constant iterator pointing to the beginning of the set, or CEnd() const if the set is empty.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*> CBegin()
const;
```

### Returns

System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to the beginning of the set, or CEnd() const if the set is empty.

### Implementation

set.cm, page 1

### CEnd() const Member Function

Returns a constant iterator pointing one past the end of the set.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*> CEnd()
const;
```

#### Returns

System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing one past the end of the set.

#### Implementation

set.cm, page 2

### CFind(const T&) const Member Function

Searches an element from the set and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*> CFind(const
T& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const T& | An element to search. |

#### Returns

System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise.

### Clear() Member Function

Makes the set empty.

### Syntax

```
public void Clear();
```

### Implementation

set.cm, page 2

### Count() const Member Function

Returns the number of elements in the set.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the set.

#### Implementation

set.cm, page 2

### End() Member Function

Returns an iterator pointing one past the end of the set.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, T&, T*> End();
```

### Returns

System.Collections.RedBlackTreeNodeIterator<T, T&, T*>

Returns an iterator pointing one past the end of the set.

### Implementation

set.cm, page 2

### End() const Member Function

Returns a constant iterator pointing one past the end of the set.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*> End() const;
```

#### Returns

System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing one past the end of the set.

### Find(const T&) Member Function

Searches an element from the set and returns an iterator pointing to it if found, or End() iterator otherwise.

### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, T&, T*> Find(const T& key);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| key | const T& | An element to seach. |

### Returns

System.Collections.RedBlackTreeNodeIterator<T, T&, T*>

Returns an iterator pointing to the found element if the search was successful, or End() iterator otherwise.

### Implementation

set.cm, page 2

### Find(const T&) const Member Function

Searches an element from the set and returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise.

#### Syntax

```
public System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*> Find(const
T& key) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const T& | An element to search. |

#### Returns

System.Collections.RedBlackTreeNodeIterator<T, const T&, const T*>

Returns a constant iterator pointing to it if found, or CEnd() const iterator otherwise.

### Insert(const T&) Member Function

Inserts an element to the set if it is not already there and returns a pair consisting of an iterator pointing to the inserted element and **true** if element was inserted, and a pair consisting of an iterator pointing to the existing element and **false** otherwise.

### Syntax

```
public System.Pair<System.Collections.RedBlackTreeNodeIterator<T, T&, T*>, bool>
Insert(const T& value);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| value | const T& | An element to insert. |

### Returns

System.Pair<System.Collections.RedBlackTreeNodeIterator<T, T&, T*>, bool>

Returns a pair consisting of an iterator pointing to the inserted element and **true** if element was inserted, and a pair consisting of an iterator pointing to the existing element and **false** otherwise.

### IsEmpty() const Member Function

Returns true if the set is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the set is empty, false otherwise.

#### Implementation

set.cm, page 2

### Remove(System.Collections.RedBlackTreeNodeIterator<T, T&, T*>) Member Function

Removes an element pointed by the given iterator from the set.

#### Syntax

```
public void Remove(System.Collections.RedBlackTreeNodeIterator<T, T&, T*> pos);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pos | System.Collections.RedBlackTreeNodeIterator<T, T&-, T*> | An iterator pointing to the element to remove. |

#### Remove(const T&) Member Function

Removes an element from the set. If the element was not found, does nothing.

#### Syntax

```
public bool Remove(const T& key);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| key | const T& | An element to remove. |

#### Returns

bool

Returns true if element was removed, false otherwise.

#### Implementation

set.cm, page 2

### 3.7.22.4 Nonmember Functions

| Function | Description |
|---|---|
| operator<(const System.Collections.Set<T, C>-&, const System.Collections.Set<T, C>&) | Compares two sets for less than relationship and returns true if the first set comes lexicographically before the second set, false otherwise. |
| operator==(const System.Collections.Set<T, C>&, const System.Collections.Set<T, C>&) | Compares two sets for equality and returns true if both contain the same number of pairwise equal elements, false otherwise. |

### operator<(const System.Collections.Set<T, C>&, const System.Collections.Set<T, C>&) Function

Compares two sets for less than relationship and returns true if the first set comes lexicographically before the second set, false otherwise.

#### Syntax

```
public bool operator<(const System.Collections.Set<T, C>& left, const System.Collections.Set<T,
C>& right);
```

## Constraint

T is Semiregular and C is Relation and C.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.Set<T, C>& | The first set. |
| right | const System.Collections.Set<T, C>& | The second set. |

#### Returns

bool

Returns true if the first set comes lexicographically before the second set, false otherwise.

#### Example

```
using System;
using System.Collections;

//  Writes:
//  !(s1 < s2) && !(s2 < s1) => s1 == s2
//  s1 < s3
//  s4 < s1

void main()
{
    Set<string> s1;
    s1.Insert("foo");
    s1.Insert("bar");
    s1.Insert("baz");
    Set<string> s2;
    s2.Insert("foo");
    s2.Insert("bar");
    s2.Insert("baz");
    if (s1 < s2)
    {
        Console.Error() << "bug" << endl();
    }
    else if (s2 < s1)
    {
        Console.Error() << "bug" << endl();
```

```
    }
    else if (s1 != s2)
    {
        Console.Error() << "bug" << endl();
    }
    else
    {
        Console.Out() << "!(s1 < s2) && !(s2 < s1) => s1 == s2" << endl();
    }
    Set<string> s3;
    s3.Insert("foo");
    s3.Insert("bar");
    s3.Insert("fluffy");
    if (s1 < s3) //    third element of s1 is less than third element of s3
    {
        Console.Out() << "s1 < s3" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
    Set<string> s4;
    s4.Insert("bar");
    s4.Insert("baz");
    if (s1 < s4)
    {
        Console.Error() << "bug" << endl();
    }
    else if (s4 < s1)   // s4[0] == s1[0] && s4[1] == s1[1], but s4 has
        fewer elements
    {
        Console.Out() << "s4 < s1" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

**Implementation**

set.cm, page 3

### operator==(const System.Collections.Set<T, C>&, const System.Collections.Set<T, C>&) Function

Compares two sets for equality and returns true if both contain the same number of pairwise equal elements, false otherwise.

#### Syntax

```
public bool operator==(const System.Collections.Set<T, C>& left, const System.Collections.Set<T
C>& right);
```

## Constraint

T is Regular and C is Relation and C.Domain is T

#### Parameters

| Name | Type | Description |
|---|---|---|
| left | const System.Collections.Set<T, C>& | The first set. |
| right | const System.Collections.Set<T, C>& | The second set. |

#### Returns

bool

Returns true if both contain the same number of pairwise equal elements, false otherwise.

#### Example

```
using System;
using System.Collections;

//  Writes:
//  s1 == s2
//  s1 != s3
//  s1 != s4

void main()
{
    Set<string> s1;
    s1.Insert("foo");
    s1.Insert("bar");
    s1.Insert("baz");
    Set<string> s2;
    s2.Insert("bar");
    s2.Insert("foo");
    s2.Insert("baz");
    if (s1 == s2)   // same number of pairwise equal elements
    {
        Console.Out() << "s1 == s2" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
```

```
    }
    Set<string> s3;
    s3.Insert("foo");
    s3.Insert("bar");
    s3.Insert("fluffy");
    if (s1 != s3) //    third element differ
    {
        Console.Out() << "s1 != s3" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
    Set<string> s4;
    s4.Insert("foo");
    s4.Insert("bar");
    if (s1 != s4)   // different number of elements
    {
        Console.Out() << "s1 != s4" << endl();
    }
    else
    {
        Console.Error() << "bug" << endl();
    }
}
```

**Implementation**

set.cm, page 2

### 3.7.23 Stack<T> Class

A last-in first-out data structure.

#### Syntax

```
public class Stack<T>;
```

#### Constraint

T is Semiregular

#### 3.7.23.1 Example

```
using System;
using System.Collections;

//  Writes:
//  foo at the top
//  bar at the top
//  baz at the top
//  baz popped, 2 items in the stack
//  bar popped, 1 items in the stack
//  foo popped, 0 items in the stack

void main()
{
    Stack<string> stack;
    stack.Push("foo");
    Console.Out() << stack.Top() << " at the top" << endl();
    stack.Push("bar");
    Console.Out() << stack.Top() << " at the top" << endl();
    stack.Push("baz");
    Console.Out() << stack.Top() << " at the top" << endl();
    while (!stack.IsEmpty())
    {
        string popped = stack.Pop();
        Console.Out() << popped << " popped, " << stack.Count() << " items
            in the stack" << endl();
    }
}
```

### 3.7.23.2 Type Definitions

| Name | Type | Description |
|---|---|---|
| ValueType | T | The type of the stack element. |

### 3.7.23.3 Member Functions

| Member Function | Description |
|---|---|
| Stack<T>() | Default constructor. Constructs an empty stack. |
| Stack<T>(const System.Collections.Stack<T>-&) | Copy constructor. |
| operator=(const System.Collections.Stack<T>-&) | Copy assignment. |
| Stack<T>(System.Collections.Stack<T>&&) | Move constructor. |
| operator=(System.Collections.Stack<T>&&) | Move assignment. |
| Clear() | Makes the stack empty. |
| Count() const | Returns the number of elements in the stack. |
| IsEmpty() const | Returns true if the stack is empty, false otherwise. |
| Pop() | Removes an element from the top of the stack and returns it. |
| Push(T&&) | Moves an element to the top of the stack. |
| Push(const T&) | Copies an element to the top of the stack. |
| Top() | Return a reference to the element at the top of the stack. |
| Top() const | Returns a constant reference to the element at the top of the stack. |

### Stack<T>() Member Function

Default constructor. Constructs an empty stack.

#### Syntax

```
public Stack<T>();
```

## Implementation

stack.cm, page 1

### Stack<T>(const System.Collections.Stack<T>&) Member Function

Copy constructor.

#### Syntax

```
public Stack<T>(const System.Collections.Stack<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Stack<T>& | Argument to copy. |

### operator=(const System.Collections.Stack<T>&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Collections.Stack<T>& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Collections.Stack<T>& | Argument to assign. |

### Stack<T>(System.Collections.Stack<T>&&) Member Function

Move constructor.

#### Syntax

```
public Stack<T>(System.Collections.Stack<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Stack<T>&& | Argument to move from. |

## Implementation

stack.cm, page 1

### operator=(System.Collections.Stack<T>&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Collections.Stack<T>&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Collections.Stack<T>&& | Argument to assign from. |

## Implementation

stack.cm, page 1

### Clear() Member Function

Makes the stack empty.

### Syntax

```
public void Clear();
```

## Implementation

stack.cm, page 2

### Count() const Member Function

Returns the number of elements in the stack.

#### Syntax

```
public int Count() const;
```

#### Returns

int

Returns the number of elements in the stack.

#### Implementation

stack.cm, page 1

### IsEmpty() const Member Function

Returns true if the stack is empty, false otherwise.

#### Syntax

```
public bool IsEmpty() const;
```

#### Returns

bool

Returns true if the stack is empty, false otherwise.

#### Implementation

stack.cm, page 1

### Pop() Member Function

Removes an element from the top of the stack and returns it.

#### Syntax

```
public T Pop();
```

#### Returns

T

Returns the removed element.

### Push(T&&) Member Function

Moves an element to the top of the stack.

#### Syntax

```
public void Push(T&& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| item | T&& | An element to push. |

### Push(const T&) Member Function

Copies an element to the top of the stack.

#### Syntax

```
public void Push(const T& item);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| item | const T& | An element to push. |

### Top() Member Function

Return a reference to the element at the top of the stack.

### Syntax

```
public T& Top();
```

### Returns

T&

Return a reference to the element at the top of the stack.

### Implementation

stack.cm, page 2

### Top() const Member Function

Returns a constant reference to the element at the top of the stack.

### Syntax

```
public const T& Top() const;
```

### Returns

const T&

Returns a constant reference to the element at the top of the stack.

### Implementation

stack.cm, page 2

## 3.8 Functions

| Function | Description |
|---|---|
| ConstructiveCopy(ValueType*, ValueType*, int) | Copies a sequence of values by constructing them into raw memory. |
| ConstructiveMove(ValueType*, ValueType*, int) | Moves a sequence of values by moving them into raw memory. |
| Destroy(ValueType*, int) | Destroys a sequence of values but does not release the memory allocated for them. |
| GetHashCode(char) | Returns a hash code for given chararacter. |
| GetHashCode(long) | Returns a hash code for the given long value. |
| GetHashCode(uchar) | Returns a hash code for the given uchar value. |
| GetHashCode(ulong) | Returns a hash code for the given ulong value. |
| GetHashCode(void*) | Returns a hash code for the given pointer. |
| GetHashCode(wchar) | Returns a hash code for the given wchar value. |

### 3.8.24 ConstructiveCopy(ValueType*, ValueType*, int) Function

Copies a sequence of values by constructing them into raw memory.

#### Syntax

```
public void ConstructiveCopy(ValueType* to, ValueType* from, int count);
```

## Constraint

ValueType is CopyConstructible

## Parameters

| Name | Type | Description |
|---|---|---|
| to | ValueType* | A pointer to beginning of raw memory to copy the elements to. |
| from | ValueType* | A pointer to elements to copy. |
| count | int | The number of elements to copy. |

## Implementation

list.cm, page 7

### 3.8.25 ConstructiveMove(ValueType*, ValueType*, int) Function

Moves a sequence of values by moving them into raw memory.

#### Syntax

```
public void ConstructiveMove(ValueType* to, ValueType* from, int count);
```

## Constraint

ValueType is MoveConstructible

## Parameters

| Name | Type | Description |
|---|---|---|
| to | ValueType* | A pointer to beginning of raw memory to move the elements to. |
| from | ValueType* | A pointer to elements to move. |
| count | int | The number of elements to move. |

## Implementation

list.cm, page 7

### 3.8.26 Destroy(ValueType*, int) Function

Destroys a sequence of values but does not release the memory allocated for them.

#### Syntax

```
public void Destroy(ValueType* items, int count);
```

#### Constraint

ValueType is Destructible

#### Parameters

| Name | Type | Description |
|---|---|---|
| items | ValueType* | A pointer to elements to destroy. |
| count | int | The number of elements to destroy. |

#### Implementation

list.cm, page 7

### 3.8.27 GetHashCode(char) Function

Returns a hash code for given chararacter.

#### Syntax

```
public ulong GetHashCode(char x);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | char | A character. |

## Returns

ulong

Returns a hash code for the given chararacter.

### 3.8.28 GetHashCode(long) Function

Returns a hash code for the given long value.

#### Syntax

```
public ulong GetHashCode(long x);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | long | A long value. |

## Returns

ulong

Returns a hash code for the given long value.

### 3.8.29 GetHashCode(uchar) Function

Returns a hash code for the given uchar value.

#### Syntax

```
public ulong GetHashCode(uchar x);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | uchar | A uchar value. |

## Returns

ulong

Returns a hash code for the given uchar value.

### 3.8.30 GetHashCode(ulong) Function

Returns a hash code for the given ulong value.

#### Syntax

```
public ulong GetHashCode(ulong x);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | ulong | A ulong value. |

## Returns

ulong

Returns a hash code for the given ulong value.

### 3.8.31 GetHashCode(void*) Function

Returns a hash code for the given pointer.

#### Syntax

```
public ulong GetHashCode(void* x);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | void* | A pointer. |

## Returns

ulong

Returns a hash code for the given pointer.

### 3.8.32 GetHashCode(wchar) Function

Returns a hash code for the given wchar value.

#### Syntax

```
public ulong GetHashCode(wchar x);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | wchar | A wchar value. |

## Returns

ulong

Returns a hash code for the given wchar value.

# 4 System.Concepts Namespace

Contains system library concepts.

Figures 4.1, 4.2, 4.3, 4.4 and 4.5 contain the concepts in this namespace.

Figure 4.1: Concept Diagram 1: Basic Concepts

Figure 4.2: Concept Diagram 2: Iterator Concepts

Figure 4.3: Concept Diagram 3: Container Concepts

Figure 4.4: Concept Diagram 4: Functional Concepts

Figure 4.5: Concept Diagram 5: Algebraic Concepts

# 4.9 Concepts

| Concept | Description |
|---|---|
| AdditiveGroup<T> | An additive group is an additive monoid that has an inverse of addition operation. |
| AdditiveMonoid<T> | An additive semigroup that has an identitity element 0 is called an additive monoid. |
| AdditiveSemigroup<T> | A set with an associative and commutative addition operation + is called an additive semigroup. |
| BackInsertionSequence<T> | A container with an **Add** member function, that adds elements to the end of the container is called a back insertion sequence. |
| BidirectionalContainer<T> | A container whose iterators are bidirectional iterators is a bidirectional container. |
| BidirectionalIterator<T> | An iterator that can be incremented and decremented is a bidirectional iterator. |
| BinaryFunction<T> | A function object that implements an **operator()** member function that accepts two parameters is called a binary function. |
| BinaryOperation<T> | A binary function whose result type is same as its first argument type is called a binary operation. |
| BinaryPredicate<T> | A binary function whose result type is **bool** is called a binary predicate. |
| CommutativeSemiring<T> | A semiring with commutative multiplication operation is called a commutative semiring. |
| Container<T> | A class that contains other objects is called a container. A container must expose the type of its contained object as an associated type named ValueType, and provide iterator types for iterating through the container, for example. |
| ConversionFromByte<T> | Implementation detail. |
| ConversionFromSByte<T> | Implementation detail. |
| CopyAssignable<T, U> | Types T and U satisfy the CopyAssignable<T, U> concept if type T has an copy assignment operator taking a parameter of type U. |

| | |
|---|---|
| CopyAssignable<T> | Copy assignable type can be target of an copy assignment operation. |
| CopyConstructible<T> | A copy constructible type can be copied. |
| Copyable<T> | A copy constructible and copy assignable type is copyable. |
| DefaultConstructible<T> | A default constructible type has a constructor that takes no parameters. |
| Destructible<T> | A destructible type has a user defined or compiler generated destructor, or is trivially destructible. |
| EqualityComparable<T, U> | Types T and U satisfy the cross-type equality comparable concept, if T is equality comparable, U is equality comparable and they have a common type that is equality comparable. |
| EqualityComparable<T> | An equality comparable type can be compared for equality and inequality. |
| EuclideanSemiring<T> | Euclidean semiring is a commutative semiring that has division and remainder operations. |
| ForwardContainer<T> | A container whose iterators are forward iterators is a forward container. |
| ForwardIterator<T> | A multipass iterator that can be incremented is a forward iterator. |
| FrontInsertionSequence<T> | A container with an **InsertFront** member function is called a front insertion sequence. |
| HashFunction<T, Key> | A unary function that computes a hash code from a key is a hash function. |
| InputIterator<T> | A one-pass iterator that can be incremented and compared for equality is an input iterator. |
| InsertionSequence<T> | A container with an **Insert** member function is called an insertion sequence. |
| Integer<I> | An integer type supports the usual integer operations. |

| | |
|---|---|
| KeySelectionFunction<T, Key, Value> | A unary function that extracts a key from a value type is called a key selection function. |
| LessThanComparable<T, U> | Types T and U satisfy the cross-type less than comparable concept, if T is less than comparable, U is less than comparable and they have a common type that is less than comparable. |
| LessThanComparable<T> | An less than comparable type can be compared for less than, greater than, less than or equal to and greater than or equal to releations. |
| Movable<T> | A move constructible and move assignable type is movable. |
| MoveAssignable<T> | A type equipped with move assignment operator is called move assignable. |
| MoveConstructible<T> | A type equipped with move constructor is called move constructible. |
| MultiplicativeGroup<T> | A multiplicative monoid with a division operation is called a multiplicative group. |
| MultiplicativeMonoid<T> | A multiplicative semigroup with an identity element is called a multiplicative monoid. |
| MultiplicativeSemigroup<T> | A set with an associative multiplication operation is called a multiplicative semigroup. |
| OrderedAdditiveGroup<T> | An ordered additive monoid that forms also an additive group is called an ordered additive group. |
| OrderedAdditiveMonoid<T> | An ordered additive semigroup that forms also an additive monoid is called an ordered additive monoid. |
| OrderedAdditiveSemigroup<T> | An additive semigroup with a total ordering relation on its elements is called an ordered additive semigroup. |
| OrderedMultiplicativeSemigroup<T> | A multiplicative semigroup with a total ordering relation on its elements is called an ordered multiplicative semigroup. |

| | |
|---|---|
| OutputIterator<T> | A writable iterator that is incrementable is an output iterator. |
| RandomAccessContainer<T> | A container whose iterators are random access iterators is a random access container. |
| RandomAccessIterator<T> | An iterator that supports incrementing, decrementing, subscripting, adding or subtracting an integer offset, and computing the difference of two iterators is a random access iterator. |
| Regular<T> | A regular type behaves like a built-in type: its objects can be default initialized, either copied, or moved (or both) and compared for equality and inequality. |
| Relation<T, U, V> | A binary predicate with two not necessarily same argument types satisfy a multiparameter relation concept. |
| Relation<T> | A binary predicate whose argument types are same is called a relation. |
| Semiregular<T> | A semiregular type behaves in many ways like a built-in type: its objects can be default initialized, either copied or moved (or both), but not necessarily compared for equality and inequality. |
| Semiring<T> | A set with addition and multiplication operations that are connected with given axioms is called a semiring. |
| SignedInteger<I> | An integer type with a conversion from **sbyte** is called a signed integer. |
| TotallyOrdered<T, U> | Types T and U satisfy the cross-type totally ordered concept if T is totally ordered, U is totally ordered and they have a common type that is totally ordered. |
| TotallyOrdered<T> | A totally ordered type is a regular type that can be also compared for less than, greater than, less than or equal to, and greater than or equal to relationships. |

| | |
|---|---|
| TrivialIterator<T> | A trivial iterator concept collects together requirements common to all iterator types. |
| UnaryFunction<T> | A function object that implements an **operator()** that accepts one parameter is called a unary function. |
| UnaryOperation<T> | A unary function whose result type and argument type are same is called a unary operation. |
| UnaryPredicate<T> | A unary function whose result type is **bool** is called a unary predicate. |
| UnsignedInteger<U> | An integer type with a conversion from **byte** is called an unsigned integer. |

### 4.9.1 AdditiveGroup<T> Concept

An additive group is an additive monoid that has an inverse of addition operation.

#### Syntax

```
public concept AdditiveGroup<T>;
```

#### Refines

System.Concepts.AdditiveMonoid<T>

#### Constraints

T operator-(T);

T operator-(T, T);

#### Axioms

```
unaryMinusIsInverseOp(T a)
{
    a + (-a) == 0 && (-a) + a == 0;
}
subtract(T a, T b)
{
    a - b == a + (-b);
}
```

#### Models

Integer and floating-point types with + and - are partial models of an additive group.

### 4.9.2 AdditiveMonoid<T> Concept

An additive semigroup that has an identitity element 0 is called an additive monoid.

#### Syntax

```
public concept AdditiveMonoid<T>;
```

#### Refines

System.Concepts.AdditiveSemigroup<T>

#### Constraints

ConversionFromSByte<T> or ConversionFromByte<T>

#### Axioms

```
zeroIsIdentityElement(T a)
{
    a + 0 == a && 0 + a == a;
}
```

#### Models

Integer and floating-point types with + are partial models of an additive monoid.

### 4.9.3 AdditiveSemigroup<T> Concept

A set with an associative and commutative addition operation + is called an additive semigroup.

#### Syntax

```
public concept AdditiveSemigroup<T>;
```

#### Constraints

T is Regular

T operator+(T, T);

#### Axioms

```
additionIsAssociative(T a, T b, T c)
{
    (a + b) + c == a + (b + c);
}
additionIsCommutative(T a, T b)
{
    a + b == b + a;
}
```

#### Models

Integer and floating-point types with + are partial models of an additive semigroup.

### 4.9.4 BackInsertionSequence<T> Concept

A container with an **Add** member function, that adds elements to the end of the container is called a back insertion sequence.

#### Syntax

```
public concept BackInsertionSequence<T>;
```

#### Refines

System.Concepts.Container<T>

#### Constraints

void T.Add(T.ValueType);

#### Models

List<T> and LinkedList<T> are a back insertion sequences.

### 4.9.5 BidirectionalContainer<T> Concept

A container whose iterators are bidirectional iterators is a bidirectional container.

#### Syntax

```
public concept BidirectionalContainer<T>;
```

#### Refines

System.Concepts.ForwardContainer<T>

#### Constraints

T.Iterator is BidirectionalIterator and T.ConstIterator is BidirectionalIterator

#### Models

List<T>, LinkedList<T>, Set<T, C> and Map<Key, Value, KeyCompare>
are bidirectional containers.

### 4.9.6 BidirectionalIterator<T> Concept

An iterator that can be incremented and decremented is a bidirectional iterator.

#### Syntax

```
public concept BidirectionalIterator<T>;
```

#### Refines

System.Concepts.ForwardIterator<T>

#### Constraints

T& operator–();

#### Models

LinkedListNodeIterator<T, R, P> and RedBlackTreeNodeIterator<T, R, P> are bidirectional iterators.

### 4.9.7 BinaryFunction<T> Concept

A function object that implements an **operator()** member function that accepts two parameters is called a binary function.

#### Syntax

```
public concept BinaryFunction<T>;
```

#### Constraints

T is Semiregular

typename T.FirstArgumentType;

typename T.SecondArgumentType;

typename T.ResultType;

T.FirstArgumentType is Semiregular and T.SecondArgumentType is Semiregular

T.ResultType operator()(T.FirstArgumentType, T.SecondArgumentType);

#### Models

Plus<T>, Minus<T>, Multiplies<T>,
Divides<T> and Remainder<T> are
binary functions.

### 4.9.8 BinaryOperation<T> Concept

A binary function whose result type is same as its first argument type is called a binary operation.

#### Syntax

```
public concept BinaryOperation<T>;
```

#### Refines

System.Concepts.BinaryFunction<T>

#### Constraints

T.ResultType is T.FirstArgumentType

#### Models

Plus<T>, Minus<T>, Multiplies<T>, Divides<T> and Remainder<T> are binary operations.

### 4.9.9 BinaryPredicate<T> Concept

A binary function whose result type is **bool** is called a binary predicate.

#### Syntax

```
public concept BinaryPredicate<T>;
```

#### Refines

System.Concepts.BinaryFunction<T>

#### Constraints

T.ResultType is T.ResultType

#### Models

EqualTo<T>, EqualTo2<T, U>, NotEqualTo<T>, NotEqualTo2<T, U>, Less<T>, Less2<T, U>, Greater<T>, Greater2<T, U>, LessOrEqualTo<T>, LessOrEqualTo2<T, U>, GreaterOrEqualTo<T>, GreaterOrEqualTo2<T, U> are binary predicates.

### 4.9.10 CommutativeSemiring<T> Concept

A semiring with commutative multiplication operation is called a commutative semiring.

#### Syntax

```
public concept CommutativeSemiring<T>;
```

#### Refines

System.Concepts.Semiring<T>

#### Axioms

```
multiplicationIsCommutative(T a, T b)
{
    a * b == b * a;
}
```

#### Models

Integer types with + and * are partial models of a commutative semiring.

### 4.9.11 Container<T> Concept

A class that contains other objects is called a container. A container must expose the type of its contained object as an associated type named ValueType, and provide iterator types for iterating through the container, for example.

#### Syntax

```
public concept Container<T>;
```

#### Constraints

T is Semiregular

typename T.ValueType;

typename T.Iterator;

typename T.ConstIterator;

T.Iterator is TrivialIterator and T.ConstIterator is TrivialIterator and T.ValueType is T.Iterator.ValueType

T.Iterator T.Begin();

T.ConstIterator T.CBegin();

T.Iterator T.End();

T.ConstIterator T.CEnd();

int T.Count();

bool T.IsEmpty();

#### Models

List<T>,
LinkedList<T>,
Set<T, C>,
Map<Key, Value, KeyCompare>,
HashSet<T, H, C>,
HashMap<K, T, H, C> and
ForwardList<T> are containers.

### 4.9.12 ConversionFromByte<T> Concept

Implementation detail.

#### Syntax

```
public concept ConversionFromByte<T>;
```

#### Constraints

T(byte);

### 4.9.13 ConversionFromSByte<T> Concept

Implementation detail.

#### Syntax

```
public concept ConversionFromSByte<T>;
```

#### Constraints

T(sbyte);

## 4.9.14 CopyAssignable<T, U> Concept

Types T and U satisfy the **CopyAssignable<T, U>** concept if type T has an copy assignment operator taking a parameter of type U.

### Syntax

```
public concept CopyAssignable<T, U>;
```

### Constraints

void operator=(const U&);

#### 4.9.14.1 Example

```
using System;
using System.Concepts;

//  Writes:
//  Error T138 in file C:/Programming/cmajor++/doc/lib/examples/System.
    Concepts.CopyAssignable.T.U.cm at line 38:
//  types (C, A) do not satisfy the requirements of concept 'System.Concepts
    .CopyAssignable' because
//  there is no member function with signature 'void C.operator=(const A&)':
//  Tester<C, A> tca;   // error
//  ^

class A
{
}

//  CopyAssignable<B, A> is true...

class B
{
    public void operator=(const A& a) {}
}

//  But CopyAssignable<C, A> is false...

class C
{
}

class Tester<T, U> where CopyAssignable<T, U>
{
}

void main()
{
    A a;
    B b;
    b = a;              // calls copy assignment operation of B
    Tester<B, A> tba;   // ok
    Tester<C, A> tca;   // error
}
```

## 4.9.15 CopyAssignable<T> Concept

Copy assignable type can be target of an copy assignment operation.

### Syntax

```
public concept CopyAssignable<T>;
```

### Constraints

void operator=(const T&);

### Models

All built-in types (**int**, **double**, **bool**, **char**, etc...) are copy assignable.
All types in the Collections namespace (List<T>, Set<T, C>, etc...) are copy assignable.
System.String, Exception, Pair<T, U> and SharedPtr<T>, for example, are also copy assignable,
but UniquePtr<T> is not copy assignable, because its copy assignment operator is suppressed.

#### 4.9.15.1 Example

```
using System;
using System.Concepts;

//  Writes:
//  Error T138 in file C:/Programming/cmajor++/doc/lib/examples/System.
    Concepts.CopyAssignable.T.cm at line 53:
//  type 'D' does not satisfy the requirements of concept 'System.Concepts.
    CopyAssignable' because
//  there is no member function with signature 'void D.operator=(const D&)':
//  Tester<D> td;   // error
//  ^

// A has compiler generated copy assignment operation:

class A
{
}

// B has compiler generated copy assignment operation:

class B
{
    public default void operator=(const B&);
}

// C has user defined copy assignment operation:

class C
{
    public void operator=(const C& that) {}
}

//  But D is not copy assignable
//  because user defined destructor
```

```
//  suppresses the compiler from generating
//  a copy assignment operation:

class D
{
    public ~D() {}
}

class Tester<T> where T is CopyAssignable
{
}

void main()
{
    A a;
    A a2;
    a = a2;          // calls copy assignment operation
    Tester<A> ta;    // ok
    Tester<B> tb;    // ok
    Tester<C> tc;    // ok
    Tester<D> td;    // error
}
```

## 4.9.16 CopyConstructible<T> Concept

A copy constructible type can be copied.

### Syntax

```
public concept CopyConstructible<T>;
```

### Constraints

T(const T&);

### Axioms

```
copyIsEqual(T a)
{
    eq(T(a), a);
}
```

### Models

All built-in types (**int**, **double**, **bool**, **char**, etc...) are copy constructible.
All types in the Collections namespace (List<T>, Set<T, C>, etc...) are copy constructible if their value type is copy constructible.
System.String, Exception, Pair<T, U> and SharedPtr<T>, for example, are also copy constructible,
but UniquePtr<T> is not copy constructible, because its copy constructor is suppressed.

#### 4.9.16.1 Example

```
using System;
using System.Concepts;

//  Writes:
//  Error T138 in file C:/Programming/cmajor++/doc/lib/examples/System.
    Concepts.CopyConstructible.cm at line 53:
//  type 'D' does not satisfy the requirements of concept 'System.Concepts.
    CopyConstructible' because
//  there is no constructor with signature 'D.D(const D&)':
//  Tester<D> td;   // error
//  ^

// A has compiler generated copy constructor:

class A
{
}

// B has compiler generated copy constructor:

class B
{
    public default B(const B&);
}
```

```
// C has user defined copy constructor:

class C
{
    public C(const C& that) {}
}

//  But D is not copy constructible
//  because user defined destructor
//  suppresses the compiler from generating
//  a copy constructor:

class D
{
    public ~D() {}
}

class Tester<T> where T is CopyConstructible
{
}

void main()
{
    A a;
    A a2(a);          // a2 is copy constructed
    A a3 = a;         // a3 is copy constructed
    Tester<A> ta;     // ok
    Tester<B> tb;     // ok
    Tester<C> tc;     // ok
    Tester<D> td;     // error
}
```

### 4.9.17 Copyable<T> Concept

A copy constructible and copy assignable type is copyable.

#### Syntax

```
public concept Copyable<T>;
```

#### Constraints

T is CopyConstructible and T is CopyAssignable

## 4.9.18 DefaultConstructible<T> Concept

A default constructible type has a constructor that takes no parameters.

### Syntax

```
public concept DefaultConstructible<T>;
```

### Constraints

T();

NonReferenceType<T>

### Models

All built-in types (**int**, **double**, **bool**, **char**, etc...) are default constructible.
All types in the Collections namespace (List<T>, Set<T, C>, etc...) are default constructible.
System.String, Exception, Pair<T, U>, UniquePtr<T> and SharedPtr<T>, for example, are also default constructible.

### Remarks

The default constructor initializes a value of its type to the natural default value: that is 0 for a numeric type, **false** for a Boolean type, empty string for a string type, empty container for a container type, etc...

#### 4.9.18.1 Example

```
using System;
using System.Concepts;

//  Writes:
//  Error T138 in file C:/Programming/cmajor++/doc/lib/examples/System.
    Concepts.DefaultConstructible.cm at line 55:
//  type 'D' does not satisfy the requirements of concept 'System.Concepts.
    DefaultConstructible' because
//  there is no constructor with signature 'D.D()':
//  Tester<D> td;   // error
//  ^

// A has compiler generated default constructor:

class A
{
}

// B has compiler generated default constructor:

class B
{
    public default B();
}

// C has user defined default constructor:
```

```
class C
{
    public C() {}
}

//  But D is not default constructible
//  because user defined constructor
//  suppresses the compiler from generating
//  a default constructor:

class D
{
    public D(int x) {}
}

class Tester<T> where T is DefaultConstructible
{
}

void main()
{
    int x;            // x is default constructed
    Tester<int> ti;   // ok
    A a;              // a is default constructed
    Tester<A> ta;     // ok
    B b;              // b is default constructed
    Tester<B> tb;     // ok
    C c;              // c is default constructed
    Tester<C> tc;     // ok
    Tester<D> td;     // error
    //D d;            // would generate error: "overload resolution failed: '
        @constructor(D*)' not found..."
}
```

## 4.9.19 Destructible<T> Concept

A destructible type has a user defined or compiler generated destructor, or is trivially destructible.

### Syntax

```
public concept Destructible<T>;
```

### Constraints

~T();

### Models

All types in Cmajor are destructible.

#### 4.9.19.1 Example

```
using System;
using System.Concepts;

// A has trivial destructor:

class A
{
}

// B has compiler generated destructor:

class B
{
    public default ~B();
}

// C has user defined destructor:

class C
{
    public ~C() {}
}

class Tester<T> where T is Destructible
{
}

void main()
{
    {
        A a;
//      ...
//      <-- a is destroyed here
    }
    Tester<A> ta;   // ok
    Tester<B> tb;   // ok
    Tester<C> tc;   // ok
}
```

### 4.9.20 EqualityComparable<T, U> Concept

Types T and U satisfy the cross-type equality comparable concept, if T is equality comparable, U is equality comparable and they have a common type that is equality comparable.

#### Syntax

```
public concept EqualityComparable<T, U>;
```

#### Refines

Common<T, U>

#### Constraints

T is EqualityComparable and U is EqualityComparable and CommonType is EqualityComparable

### 4.9.21 EqualityComparable<T> Concept

An equality comparable type can be compared for equality and inequality.

#### Syntax

```
public concept EqualityComparable<T>;
```

#### Constraints

bool operator==(T, T);

#### Axioms

```
equal(T a, T b)
{
    a == b <=> eq(a, b);
}
reflexive(T a)
{
    a == a;
}
symmetric(T a, T b)
{
    a == b => b == a;
}
transitive(T a, T b, T c)
{
    a == b && b == c => a == c;
}
notEqualTo(T a, T b)
{
    a != b <=> !(a == b);
}
```

### 4.9.22 EuclideanSemiring<T> Concept

Euclidean semiring is a commutative semiring that has division and remainder operations.

#### Syntax

```
public concept EuclideanSemiring<T>;
```

#### Refines

System.Concepts.CommutativeSemiring<T>

#### Constraints

T operator%(T, T);

T operator/(T, T);

#### Axioms

```
quotientAndRemainder(T a, T b)
{
    b != 0 => a == a / b * b + a % b;
}
```

### 4.9.23 ForwardContainer<T> Concept

A container whose iterators are forward iterators is a forward container.

#### Syntax

```
public concept ForwardContainer<T>;
```

#### Refines

System.Concepts.Container<T>

#### Constraints

T.Iterator is ForwardIterator and T.ConstIterator is ForwardIterator

#### Models

ForwardList<T> is a proper forward container.
All containers in the Collections namespace are also forward containers.

### 4.9.24 ForwardIterator<T> Concept

A multipass iterator that can be incremented is a forward iterator.

#### Syntax

```
public concept ForwardIterator<T>;
```

#### Refines

System.Concepts.InputIterator<T>

#### Constraints

T is OutputIterator

#### Models

ForwardListNodeIterator<T, R, P> is a proper forward iterator.
All Cmajor iterators are also forward iterators.

### 4.9.25 FrontInsertionSequence<T> Concept

A container with an **InsertFront** member function is called a front insertion sequence.

#### Syntax

```
public concept FrontInsertionSequence<T>;
```

#### Refines

System.Concepts.Container<T>

#### Constraints

T.Iterator T.InsertFront(T.ValueType);

#### Models

List<T>, LinkedList<T> and ForwardList<T> are front insertion sequences.

### 4.9.26 HashFunction<T, Key> Concept

A unary function that computes a hash code from a key is a hash function.

#### Syntax

```
public concept HashFunction<T, Key>;
```

#### Refines

System.Concepts.UnaryFunction<T>

#### Constraints

ulong and T.ResultType is T.ResultType

#### Models

Hasher<T> is a hash function.

### 4.9.27 InputIterator<T> Concept

A one-pass iterator that can be incremented and compared for equality is an input iterator.

#### Syntax

```
public concept InputIterator<T>;
```

#### Refines

System.Concepts.TrivialIterator<T>

#### Constraints

T& operator++();

T is Regular

### 4.9.28 InsertionSequence<T> Concept

A container with an **Insert** member function is called an insertion sequence.

#### Syntax

```
public concept InsertionSequence<T>;
```

#### Refines

System.Concepts.ForwardContainer<T>

#### Constraints

T.Iterator T.Insert(T.Iterator, T.ValueType);

#### Models

List<T> and LinkedList<T> are insertion sequences.

### 4.9.29 Integer<I> Concept

An integer type supports the usual integer operations.

#### Syntax

```
public concept Integer<I>;
```

#### Constraints

I is TotallyOrdered

I operator-(I);

I operator~(I);

I& operator++(I&);

I& operator–(I&);

I operator+(I, I);

I operator-(I, I);

I operator*(I, I);

I operator/(I, I);

I operator%(I, I);

I operator<<(I, I);

I operator>>(I, I);

I operator&(I, I);

I operator|(I, I);

I operator^(I, I);

#### Models

All Cmajor integer types.

### 4.9.30 KeySelectionFunction<T, Key, Value> Concept

A unary function that extracts a key from a value type is called a key selection function.

#### Syntax

```
public concept KeySelectionFunction<T, Key, Value>;
```

#### Refines

System.Concepts.UnaryFunction<T>

#### Constraints

T.ArgumentType is Value and T.ResultType is Key

#### Models

SelectFirst<T, U> is a key selection function.

### 4.9.31 LessThanComparable<T, U> Concept

Types T and U satisfy the cross-type less than comparable concept, if T is less than comparable, U is less than comparable and they have a common type that is less than comparable.

#### Syntax

```
public concept LessThanComparable<T, U>;
```

#### Refines

Common<T, U>

#### Constraints

T is LessThanComparable and U is LessThanComparable and CommonType is LessThanComparable

### 4.9.32 LessThanComparable<T> Concept

An less than comparable type can be compared for less than, greater than, less than or equal to and greater than or equal to releations.

#### Syntax

```
public concept LessThanComparable<T>;
```

#### Constraints

bool operator<(T, T);

#### Axioms

```
irreflexive(T a)
{
    !(a < a);
}
antisymmetric(T a, T b)
{
    a < b => !(b < a);
}
transitive(T a, T b, T c)
{
    a < b && b < c => a < c;
}
total(T a, T b)
{
    a < b || a == b || a > b;
}
greaterThan(T a, T b)
{
    a > b <=> b < a;
}
greaterThanOrEqualTo(T a, T b)
{
    a >= b <=> !(a < b);
}
lessThanOrEqualTo(T a, T b)
{
    a <= b <=> !(b < a);
}
```

### 4.9.33 Movable<T> Concept

A move constructible and move assignable type is movable.

#### Syntax

```
public concept Movable<T>;
```

#### Constraints

T is MoveConstructible and T is MoveAssignable

### 4.9.34 MoveAssignable<T> Concept

A type equipped with move assignment operator is called move assignable.

#### Syntax

```
public concept MoveAssignable<T>;
```

#### Constraints

void operator=(T&&);

### 4.9.35 MoveConstructible<T> Concept

A type equipped with move constructor is called move constructible.

#### Syntax

```
public concept MoveConstructible<T>;
```

#### Constraints

T(T&&);

### 4.9.36 MultiplicativeGroup<T> Concept

A multiplicative monoid with a division operation is called a multiplicative group.

#### Syntax

```
public concept MultiplicativeGroup<T>;
```

#### Refines

System.Concepts.MultiplicativeMonoid<T>

#### Constraints

T operator/(T, T);

#### Axioms

```
multiplicativeInverseIsInverseOp(T a)
{
    a * (1/a) == 1 && (1/a) * a == 1;
}
division(T a, T b)
{
    a / b == a * (1/b);
}
```

### 4.9.37 MultiplicativeMonoid<T> Concept

A multiplicative semigroup with an identity element is called a multiplicative monoid.

#### Syntax

```
public concept MultiplicativeMonoid<T>;
```

#### Refines

System.Concepts.MultiplicativeSemigroup<T>

#### Constraints

ConversionFromSByte<T> or ConversionFromByte<T>

#### Axioms

```
oneIsIdentityElement(T a)
{
    a * 1 == a && 1 * a == a;
}
```

### 4.9.38 MultiplicativeSemigroup<T> Concept

A set with an associative multiplication operation is called a multiplicative semigroup.

#### Syntax

```
public concept MultiplicativeSemigroup<T>;
```

#### Constraints

T is Regular

T operator*(T, T);

#### Axioms

```
multiplicationIsAssociative(T a, T b, T c)
{
    (a * b) * c == a * (b * c);
}
```

### 4.9.39 OrderedAdditiveGroup<T> Concept

An ordered additive monoid that forms also an additive group is called an ordered additive group.

#### Syntax

```
public concept OrderedAdditiveGroup<T>;
```

#### Refines

System.Concepts.OrderedAdditiveMonoid<T>

#### Constraints

T is AdditiveGroup

### 4.9.40 OrderedAdditiveMonoid<T> Concept

An ordered additive semigroup that forms also an additive monoid is called an ordered additive monoid.

#### Syntax

```
public concept OrderedAdditiveMonoid<T>;
```

#### Refines

System.Concepts.OrderedAdditiveSemigroup<T>

#### Constraints

T is AdditiveMonoid

### 4.9.41 OrderedAdditiveSemigroup<T> Concept

An additive semigroup with a total ordering relation on its elements is called an ordered additive semigroup.

#### Syntax

```
public concept OrderedAdditiveSemigroup<T>;
```

#### Refines

System.Concepts.AdditiveSemigroup<T>

#### Constraints

T is TotallyOrdered

#### Axioms

```
additionPreservesOrder(T a, T b, T c)
{
    a < b => a + c < b + c;
}
```

### 4.9.42 OrderedMultiplicativeSemigroup<T> Concept

A multiplicative semigroup with a total ordering relation on its elements is called an ordered multiplicative semigroup.

#### Syntax

```
public concept OrderedMultiplicativeSemigroup<T>;
```

#### Refines

System.Concepts.MultiplicativeSemigroup<T>

#### Constraints

T is TotallyOrdered

### 4.9.43 OutputIterator<T> Concept

A writable iterator that is incrementable is an output iterator.

#### Syntax

```
public concept OutputIterator<T>;
```

#### Refines

System.Concepts.TrivialIterator<T>

#### Constraints

T& operator++();

### 4.9.44 RandomAccessContainer<T> Concept

A container whose iterators are random access iterators is a random access container.

#### Syntax

```
public concept RandomAccessContainer<T>;
```

#### Refines

System.Concepts.BidirectionalContainer<T>

#### Constraints

T.Iterator is RandomAccessIterator and T.ConstIterator is RandomAccessIterator

#### Models

List<T> is a random access container.

### 4.9.45 RandomAccessIterator<T> Concept

An iterator that supports incrementing, decrementing, subscripting, adding or subtracting an integer offset, and computing the difference of two iterators is a random access iterator.

#### Syntax

```
public concept RandomAccessIterator<T>;
```

#### Refines

System.Concepts.BidirectionalIterator<T>

#### Constraints

T.ReferenceType operator[](int index);

T operator+(T, int);

T operator+(int, T);

T operator-(T, int);

int operator-(T, T);

T is LessThanComparable

#### Models

RandomAccessIter<T, R, P> is a random access iterator.

## 4.9.46 Regular<T> Concept

A regular type behaves like a built-in type: its objects can be default initialized, either copied, or moved (or both) and compared for equality and inequality.

### Syntax

```
public concept Regular<T>;
```

### Refines

System.Concepts.Semiregular<T>

### Constraints

T is EqualityComparable

### Models

All built-in types (**int**, **double**, **bool**, **char**, etc...) are regular.
All types in the Collections namespace (List<T>, Set<T, C>, etc...) are regular, if their value type is regular.

### Remarks

If a type implements the == operator, the compiler implements the != operator automatically.

#### 4.9.46.1 Example

```
using System;
using System.Concepts;

//  Writes:
//  Error T138 in file C:/Programming/cmajor++/doc/lib/examples/System.
    Concepts.Regular.cm at line 98:
//  type 'D' does not satisfy the requirements of concept 'System.Concepts.
    Regular'
//  because type 'D' does not satisfy the requirements of concept 'System.
    Concepts.EqualityComparable' because
//  there is no member function with signature 'bool D.operator==(D)' and no
     function with signature 'bool operator==(D, D)':
//  Tester<D> td;   // error
//  ^

// A has user-defined default constructor, compiler generated copy
    constructor, copy assignment,
// move constructor and move assignment,
// it is trivially destructible and its objects can be compare for equality,
     so it is regular:

class A
{
    public A(): id(0)
    {
    }
    public A(int id_): id(id_)
```

```
    {
    }
    public int Id() const
    {
        return id;
    }
    private int id;
}

public bool operator==(const A& left, const A& right)
{
    return left.Id() == right.Id();
}

// B has user-defined default constructor, compiler generated copy
    constructor, copy assignment,
// move constructor, move assignment and destructor, and its objects can be
    compared for equality, so it is regular:

class B
{
    public B(): id(0)
    {
    }
    public B(int id_): id(id_)
    {
    }
    public default B(const B&);
    public default void operator=(const B&);
    public default B(B&&);
    public default void operator=(B&&);
    public default ~B();
    public int Id() const
    {
        return id;
    }
    private int id;
}

public bool operator==(const B& left, const B& right)
{
    return left.Id() == right.Id();
}

// C has user defined default constructor, copy constructor, copy assignment
    ,
// move constructior, move assignment and destructor,
// its objects can be compared for equality, so it is regular:

class C
{
    public C(): id(0) {}
    public C(const C& that): id(that.id) {}
    public void operator=(const C& that) { id = that.id; }
    public C(C&& that): id(that.id)
    {
        that.id = 0;
    }
```

```
    public void operator=(C&& that)
    {
        Swap(id, that.id);
    }
    public ~C() {}
    public int Id() const
    {
        return id;
    }
    private int id;
}

public bool operator==(const C& left, const C& right)
{
    return left.Id() == right.Id();
}

//  But D is not regular,
//  because its objects cannot be compared for equality:

class D
{
}

class Tester<T> where T is Regular
{
}

void main()
{
    A a;
    A a2(a);          // a2 is copy constructed
    A a3;
    a3 = a2;          // a3 is copy assigned
    Tester<A> ta;     // ok
    Tester<B> tb;     // ok
    Tester<C> tc;     // ok
    Tester<D> td;     // error
}
```

### 4.9.47 Relation<T, U, V> Concept

A binary predicate with two not necessarily same argument types satisfy a multiparameter relation concept.

#### Syntax

```
public concept Relation<T, U, V>;
```

#### Refines

System.Concepts.BinaryPredicate<T>

#### Constraints

T.FirstArgumentType is U and T.SecondArgumentType is V

### 4.9.48 Relation<T> Concept

A binary predicate whose argument types are same is called a relation.

#### Syntax

```
public concept Relation<T>;
```

#### Refines

System.Concepts.BinaryPredicate<T>

#### Constraints

typename T.Domain;

Same<T.Domain, T.FirstArgumentType> and Same<T.SecondArgumentType, T.Domain>

## 4.9.49 Semiregular<T> Concept

A semiregular type behaves in many ways like a built-in type: its objects can be default initialized, either copied or moved (or both), but not necessarily compared for equality and inequality.

### Syntax

```
public concept Semiregular<T>;
```

### Constraints

T is DefaultConstructible and T is Copyable or T is Movable and T is Destructible

### Models

All built-in types (**int**, **double**, **bool**, **char**, etc...) are semiregular.
All types in the Collections namespace (List<T>, Set<T, C>, etc...) are semiregular.
System.String, Exception, Pair<T, U> and SharedPtr<T>, for example, are also semiregular.
Although UniquePtr<T> is not copyable because its copy constructor and assignment operator are suppressed,
it is movable because it implements move constructor and move assignment operator, so it is semiregular.

### Remarks

Many containers require that the type of the contained object is semiregular.

#### 4.9.49.1 Example

```
using System;
using System.Concepts;

//  Writes:
//  Error T138 in file C:/Programming/cmajor++/doc/lib/examples/System.
    Concepts.Semiregular.cm at line 65:
//  type 'D' does not satisfy the requirements of concept 'System.Concepts.
    Semiregular' because
//  type 'D' does not satisfy the requirements of concept 'System.Concepts.
    CopyConstructible' because
//  there is no constructor with signature 'D.D(const D&)':
//  Tester<D> td;   // error
//  ^

// A has compiler generated default constructor, copy constructor and copy
    assignment,
// move constructor and move assignment and it is trivially destructible, so
     it is semiregular:

class A
{
}

// B has compiler generated default constructor, copy constructor, copy
    assignment,
// move constructor, move assignment and destructor, so it is semiregular:
```

```
class B
{
    public default B();
    public default B(const B&);
    public default void operator=(const B&);
    public default B(B&&);
    public default void operator=(B&&);
    public default ~B();
}

// C has user defined default constructor, copy constructor, copy assignment
    ,
// move constructor, move assignment and destructor, so it is semiregular:

class C
{
    public C() {}
    public C(const C& that) {}
    public void operator=(const C& that) {}
    public C(C&& that) {}
    public void operator=(C&& that) {}
    public ~C() {}
}

//  But D is not semiregular,
//  because the copy constructor and the
//  copy assignment operator are suppressed.

class D
{
    public D() {}
    suppress D(const D&);
    suppress void operator=(const D&);
}

class Tester<T> where T is Semiregular
{
}

void main()
{
    A a;
    A a2(a);          // a2 is copy constructed
    A a3;
    a3 = a2;          // a3 is copy assigned
    A a4;
    a4 = Rvalue(a3);// a4 is move assigned
    Tester<A> ta;    // ok
    Tester<B> tb;    // ok
    Tester<C> tc;    // ok
    Tester<D> td;    // error
}
```

### 4.9.50 Semiring<T> Concept

A set with addition and multiplication operations that are connected with given axioms is called a semiring.

#### Syntax

```
public concept Semiring<T>;
```

#### Refines

System.Concepts.AdditiveMonoid<T>

#### Constraints

T is MultiplicativeMonoid

#### Axioms

```
zeroIsNotOne
{
    0 != 1;
}
multiplyingByZeroYieldsZero(T a)
{
    0 * a == 0 && a * 0 == 0;
}
distributivity(T a, T b, T c)
{
    a * (b + c) == a * b + a * c && (b + c) * a == b * a + c * a;
}
```

### 4.9.51 SignedInteger<I> Concept

An integer type with a conversion from **sbyte** is called a signed integer.

#### Syntax

```
public concept SignedInteger<I>;
```

#### Constraints

I is Integer

I(sbyte);

#### Models

**sbyte**, **short**, **int** and **long** are signed integer types.

### 4.9.52 TotallyOrdered<T, U> Concept

Types T and U satisfy the cross-type totally ordered concept if T is totally ordered, U is totally ordered and they have a common type that is totally ordered.

#### Syntax

```
public concept TotallyOrdered<T, U>;
```

#### Refines

Common<T, U>

#### Constraints

T is TotallyOrdered and U is TotallyOrdered and CommonType is TotallyOrdered

## 4.9.53 TotallyOrdered<T> Concept

A totally ordered type is a regular type that can be also compared for less than, greater than, less than or equal to, and greater than or equal to relationships.

### Syntax

```
public concept TotallyOrdered<T>;
```

### Refines

System.Concepts.Regular<T>

### Constraints

T is LessThanComparable

### Models

Arithmetic and character types (**int**, **double**, **char**, etc...) are totally ordered.
List<T>, LinkedList<T>, Set<T, C> and Map<Key, Value, KeyCompare> are totally ordered if their value type is totally ordered.

### Remarks

If a type implements the < operator, the compiler implements the >, <= and >= operators automatically.

#### 4.9.53.1 Example

```
using System;
using System.Concepts;

//  Writes:
//  Error T138 in file C:/Programming/cmajor++/doc/lib/examples/System.
    Concepts.TotallyOrdered.cm at line 127:
//  type 'D' does not satisfy the requirements of concept 'System.Concepts.
    TotallyOrdered' because
//  type 'D' does not satisfy the requirements of concept 'System.Concepts.
    LessThanComparable' because
//  there is no member function with signature 'bool D.operator<(D)' and no
    function with signature 'bool operator<(D, D)':
//  Tester<D> td;   // error
//  ^

// A has user-defined default constructor, compiler generated copy
    constructor and copy assignment,
// it is trivially destructible and its objects can be compare for equality
    and less than relationship,
// so it is totally ordered:

class A
{
    public A(): id(0)
    {
    }
```

```
    public A(int id_): id(id_)
    {
    }
    public int Id() const
    {
        return id;
    }
    private int id;
}

public bool operator==(const A& left, const A& right)
{
    return left.Id() == right.Id();
}

public bool operator<(const A& left, const A& right)
{
    return left.Id() < right.Id();
}

// B has user-defined default constructor, compiler generated copy
    constructor, copy assignment and destructor, and
// its objects can be compared for equality and less than relationship, so
    it is totally ordered:

class B
{
    public B(): id(0)
    {
    }
    public B(int id_): id(id_)
    {
    }
    public default B(const B&);
    public default void operator=(const B&);
    public default ~B();
    public int Id() const
    {
        return id;
    }
    private int id;
}

public bool operator==(const B& left, const B& right)
{
    return left.Id() == right.Id();
}

public bool operator<(const B& left, const B& right)
{
    return left.Id() < right.Id();
}

// C has user defined default constructor, copy constructor, copy assignment
// and destructor, its objects can be compared for equality and less than
    relationship,
// so it is totally ordered:
```

```
class C
{
    public C(): id(0) {}
    public C(const C& that): id(that.id) {}
    public void operator=(const C& that) { id = that.id; }
    public ~C() {}
    public int Id() const
    {
        return id;
    }
    private int id;
}

public bool operator==(const C& left, const C& right)
{
    return left.Id() == right.Id();
}

public bool operator<(const C& left, const C& right)
{
    return left.Id() < right.Id();
}

//  But D is not totally ordered,
//  because its objects cannot be compared for less than relationship:

class D
{
    public D(): id(0) {}
    public D(const D& that): id(that.id) {}
    public void operator=(const D& that) { id = that.id; }
    public ~D() {}
    public int Id() const
    {
        return id;
    }
    private int id;
}

public bool operator==(const D& left, const D& right)
{
    return left.Id() == right.Id();
}

class Tester<T> where T is TotallyOrdered
{
}

void main()
{
    A a;
    A a2(a);        // a2 is copy constructed
    A a3;
    a3 = a2;        // a3 is copy assigned
    Tester<A> ta;   // ok
    Tester<B> tb;   // ok
    Tester<C> tc;   // ok
    Tester<D> td;   // error
```

```
}
```

### 4.9.54 TrivialIterator<T> Concept

A trivial iterator concept collects together requirements common to all iterator types.

#### Syntax

```
public concept TrivialIterator<T>;
```

#### Constraints

T is Semiregular

typename T.ValueType;

T.ValueType is Semiregular

typename T.ReferenceType;

T.ReferenceType is ValueType&

T.ReferenceType operator*();

typename T.PointerType;

T.PointerType is ValueType*

T.PointerType operator->();

### 4.9.55 UnaryFunction<T> Concept

A function object that implements an **operator()** that accepts one parameter is called a unary function.

#### Syntax

```
public concept UnaryFunction<T>;
```

#### Constraints

T is Semiregular

typename T.ArgumentType;

typename T.ResultType;

T.ArgumentType is Semiregular

T.ResultType operator()(T.ArgumentType);

#### Models

Negate<T> and Identity<T> are unary functions.

### 4.9.56 UnaryOperation<T> Concept

A unary function whose result type and argument type are same is called a unary operation.

#### Syntax

```
public concept UnaryOperation<T>;
```

#### Refines

System.Concepts.UnaryFunction<T>

#### Constraints

T.ResultType is T.ArgumentType

#### Models

Negate<T> and Identity<T> are unary operations.

### 4.9.57 UnaryPredicate<T> Concept

A unary function whose result type is **bool** is called a unary predicate.

#### Syntax

```
public concept UnaryPredicate<T>;
```

#### Refines

System.Concepts.UnaryFunction<T>

#### Constraints

T.ResultType is T.ResultType

### 4.9.58 UnsignedInteger<U> Concept

An integer type with a conversion from **byte** is called an unsigned integer.

#### Syntax

```
public concept UnsignedInteger<U>;
```

#### Constraints

U is Integer

U(byte);

#### Models

**byte**, **ushort**, **uint** and **ulong** are unsigned integer types.

# 5 System.IO Namespace

Contains classes and functions for doing input and output.

Figure 5.1 contains the classes in this namespace.

Figure 5.1: Class Diagram: I/O Classes

## 5.10 Classes

| Class | Description |
|---|---|
| BinaryFileStream | A stream of bytes connected to a file. |
| ByteStream | Abstract base class for byte stream classes. Represents a stream of bytes. Byte stream can be read from or written to. |
| CloseFileException | An exception thrown when closing a file fails. |
| FileByteStream | Represents a stream of bytes connected to a file. The stream can be read from or written to but not both. |
| FileMapping | Represents memory mapped input file. |
| FileMappingFailure | Exception thrown when opening a memory mapped file fails. |
| IOBuffer | A handle to dynamically allocated memory. |
| IOException | An exception thrown when an I/O operation fails. |
| InputFileStream | A stream of characters connected to an input file. |
| InputStream | An abstract base class for input stream classes. |
| InputStringStream | A class for reading from a string. |
| InvalidPathException | An exception class that is thrown if a path contains too many .. components. |
| MemoryByteStream | Represents a stream of bytes in memory. Writing to the stream writes data to the end of the stream. Reading from the stream reads data starting from the beginning of the stream. |
| OpenFileException | An exception class thrown if opening a file fails. |
| OutputFileStream | A stream of characters connected to an output file. |
| OutputStream | An abstract base class for output stream classes. |
| OutputStringStream | A class for writing to a string. |
| Path | A static class for manipulating paths. |

## 5.10.1 BinaryFileStream Class

A stream of bytes connected to a file.

### Syntax

```
public class BinaryFileStream;
```

#### 5.10.1.1 Member Functions

| Member Function | Description |
|---|---|
| BinaryFileStream(System.IO.-BinaryFileStream&&) | Move constructor. |
| operator=(System.IO.BinaryFileStream&&) | Move constructor. |
| Close() | If the binary file stream is connected to an open file, closes the file, otherwise throws CloseFileException. |
| GetFileSize() | Returns the size of the file binary file stream is connected to. |
| Read(void*, ulong) | Reads size bytes from binary file stream into buffer and returns the number of bytes read. |
| ReadBool() | Reads a Boolean value from the binary file stream and returns it. |
| ReadByte() | Reads a **byte** from the binary file stream and returns it. |
| ReadChar() | Reads a character from the binary file stream and returns it. |
| ReadDouble() | Reads a **double** from the binary file stream and returns it. |
| ReadFloat() | Reads a **float** from the binary file stream and returns it. |
| ReadInt() | Reads an **int** from the binary file stream and returns it. |
| ReadLong() | Reads an **long** from the binary file stream and returns it. |
| ReadSByte() | Reads an **sbyte** from the binary file stream and returns it. |

| | |
|---|---|
| ReadShort() | Reads a **short** from the binary file stream and returns it. |
| ReadSize(void*, ulong) | Reads exactly size bytes from the binary file stream into buffer. If the number of bytes read is not equal to size throws an IOException. |
| ReadString() | Reads the length of a string (an **int**) followed by the contents of the string from the binary file stream and returns the string. |
| ReadUInt() | Reads a **uint** from the binary file stream and returns it. |
| ReadULong() | Reads a **ulong** from the binary file stream and returns it. |
| ReadUShort() | Reads a **ushort** from the binary file stream and returns it. |
| Seek(long, int) | Sets the current file position. |
| Tell() | Returns the current file position. |
| Write(bool) | Writes a Boolean value to the binary file stream. |
| Write(byte) | Writes a **byte** to the binary file stream. |
| Write(char) | Writes a character to the binary file stream. |
| Write(const char*) | Writes a length (an **int**) of the given C-style string followed by the contents of the C-style string to the binary file stream. |
| Write(double) | Writes a **double** to the binary file stream. |
| Write(float) | Writes a **float** to the binary file stream. |
| Write(int) | Writes an **int** to the binary file stream. |
| Write(long) | Writes a **long** to the binary file stream. |
| Write(sbyte) | Writes an **sbyte** to the binary file stream. |
| Write(short) | Writes a **short** to the binary file stream. |
| Write(uint) | Writes a **uint** to the binary file stream. |

| | |
|---|---|
| Write(ulong) | Writes a **ulong** to the binary file stream. |
| Write(ushort) | Writes a **ushort** to the binary file stream. |
| Write(void*, ulong) | Writes size bytes from buffer to the binary file stream. If the number of bytes written is not equal to size throws an IOException. |
| ~BinaryFileStream() | If the binary file stream is connected to an open file, closes the file. |

### BinaryFileStream(System.IO.BinaryFileStream&&) Member Function

Move constructor.

#### Syntax

```
public BinaryFileStream(System.IO.BinaryFileStream&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.BinaryFileStream&& | A binary file stream to move from. |

### operator=(System.IO.BinaryFileStream&&) Member Function

Move constructor.

#### Syntax

```
public void operator=(System.IO.BinaryFileStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.BinaryFileStream&& | A binary file stream to move from. |

### Close() Member Function

If the binary file stream is connected to an open file, closes the file, otherwise throws CloseFileException.

### Syntax

```
public void Close();
```

### GetFileSize() Member Function

Returns the size of the file binary file stream is connected to.

#### Syntax

```
public long GetFileSize();
```

#### Returns

long

Returns the size of the file binary file stream is connected to.

### Read(void*, ulong) Member Function

Reads size bytes from binary file stream into buffer and returns the number of bytes read.

#### Syntax

```
public int Read(void* buffer, ulong size);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buffer | void* | A buffer to read to. |
| size | ulong | Number of bytes to read. |

#### Returns

int

Returns the number of bytes read.

### ReadBool() Member Function

Reads a Boolean value from the binary file stream and returns it.

#### Syntax

```
public bool ReadBool();
```

#### Returns

bool

Returns the read Boolean value.

### ReadByte() Member Function

Reads a **byte** from the binary file stream and returns it.

### Syntax

```
public byte ReadByte();
```

### Returns

byte

Returns the read **byte**.

### ReadChar() Member Function

Reads a character from the binary file stream and returns it.

#### Syntax

```
public char ReadChar();
```

#### Returns

char

Returns the read character.

### ReadDouble() Member Function

Reads a **double** from the binary file stream and returns it.

#### Syntax

```
public double ReadDouble();
```

#### Returns

double

Returns the read **double**.

### ReadFloat() Member Function

Reads a **float** from the binary file stream and returns it.

#### Syntax

```
public float ReadFloat();
```

#### Returns

float

Returns the read **float**.

### ReadInt() Member Function

Reads an **int** from the binary file stream and returns it.

#### Syntax

```
public int ReadInt();
```

#### Returns

int

Returns the read **int**.

### ReadLong() Member Function

Reads an **long** from the binary file stream and returns it.

### Syntax

```
public long ReadLong();
```

### Returns

long

Returns the read **long**.

### ReadSByte() Member Function

Reads an **sbyte** from the binary file stream and returns it.

#### Syntax

```
public sbyte ReadSByte();
```

#### Returns

sbyte

Returns the read **sbyte**.

### ReadShort() Member Function

Reads a **short** from the binary file stream and returns it.

#### Syntax

```
public short ReadShort();
```

#### Returns

short

Returns the read **short**.

### ReadSize(void*, ulong) Member Function

Reads exactly size bytes from the binary file stream into buffer. If the number of bytes read is not equal to size throws an IOException.

#### Syntax

```
public void ReadSize(void* buffer, ulong size);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buffer | void* | A buffer to read to. |
| size | ulong | Number of bytes to read. |

### ReadString() Member Function

Reads the length of a string (an **int**) followed by the contents of the string from the binary file stream and returns the string.

#### Syntax

```
public System.String<char> ReadString();
```

#### Returns

System.String<char>

Returns the read string.

### ReadUInt() Member Function

Reads a **uint** from the binary file stream and returns it.

#### Syntax

```
public uint ReadUInt();
```

#### Returns

uint

Returns the read **uint**.

### ReadULong() Member Function

Reads a **ulong** from the binary file stream and returns it.

#### Syntax

```
public ulong ReadULong();
```

#### Returns

ulong

Returns the read **ulong**.

### ReadUShort() Member Function

Reads a **ushort** from the binary file stream and returns it.

#### Syntax

```
public ushort ReadUShort();
```

#### Returns

ushort

Returns the read **ushort**.

### Seek(long, int) Member Function

Sets the current file position.

#### Syntax

```
public long Seek(long offset, int origin);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| offset | long | An offset. |
| origin | int | One of the constants SEEK_SET, SEEK_CUR, and SEEK_END. |

#### Returns

long

Returns the current file position.

### Tell() Member Function

Returns the current file position.

### Syntax

```
public long Tell();
```

### Returns

long

Returns the current file position.

### Write(bool) Member Function

Writes a Boolean value to the binary file stream.

#### Syntax

```
public void Write(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean value to write. |

### Write(byte) Member Function

Writes a **byte** to the binary file stream.

### Syntax

```
public void Write(byte b);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A **byte** to write. |

### Write(char) Member Function

Writes a character to the binary file stream.

#### Syntax

```
public void Write(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### Write(const char*) Member Function

Writes a length (an **int**) of the given C-style string followed by the contents of the C-style string to the binary file stream.

#### Syntax

```
public void Write(const char* s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### Write(double) Member Function

Writes a **double** to the binary file stream.

#### Syntax

```
public void Write(double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A **double** to write. |

### Write(float) Member Function

Writes a **float** to the binary file stream.

#### Syntax

```
public void Write(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A **float** to write. |

### Write(int) Member Function

Writes an **int** to the binary file stream.

### Syntax

```
public void Write(int i);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An **int** to write. |

### Write(long) Member Function

Writes a **long** to the binary file stream.

#### Syntax

```
public void Write(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A **long** to write. |

### Write(sbyte) Member Function

Writes an **sbyte** to the binary file stream.

#### Syntax

```
public void Write(sbyte s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | sbyte | An **sbyte** to write. |

### Write(short) Member Function

Writes a **short** to the binary file stream.

#### Syntax

```
public void Write(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A **short** to write. |

### Write(uint) Member Function

Writes a **uint** to the binary file stream.

#### Syntax

```
public void Write(uint u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | uint | A **uint** to write. |

### Write(ulong) Member Function

Writes a **ulong** to the binary file stream.

#### Syntax

```
public void Write(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | A **ulong** to write. |

### Write(ushort) Member Function

Writes a **ushort** to the binary file stream.

#### Syntax

```
public void Write(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | A **ushort** to write. |

### Write(void*, ulong) Member Function

Writes size bytes from buffer to the binary file stream. If the number of bytes written is not equal to size throws an IOException.

#### Syntax

```
public void Write(void* buffer, ulong size);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buffer | void* | A buffer to write from. |
| size | ulong | Number of bytes to write. |

### ~BinaryFileStream() Member Function

If the binary file stream is connected to an open file, closes the file.

#### Syntax

```
public ~BinaryFileStream();
```

## 5.10.2 ByteStream Class

Abstract base class for byte stream classes. Represents a stream of bytes. Byte stream can be read from or written to.

### Syntax

```
public abstract class ByteStream;
```

### 5.10.2.1 Member Functions

| Member Function | Description |
|---|---|
| ByteStream() | Default constructor. |
| ByteStream(System.IO.ByteStream&&) | Move constructor. |
| operator=(System.IO.ByteStream&&) | Move assignment. |
| CopyTo(System.IO.ByteStream&) | Copies this byte stream to another. Uses default buffer size 16K for copying. |
| CopyTo(System.IO.ByteStream&, int) | Copies this byte stream to another using given buffer size. |
| Read(byte*, int) | Abstract member function that derived byte stream class overrides. Reads at most given number of bytes from the byte stream to the given buffer. Returns number of bytes read that might be less than the number of bytes requested. |
| ReadByte() | Abstract member function that derived byte stream class overrides. Reads one byte from the byte stream. Returns the byte read or -1 if end of stream is encountered. |
| Write(byte) | Abstract member function that derived byte stream class overrides. Writes one byte of data to the byte stream. |
| Write(byte*, int) | Abstract member function that derived byte stream class overrides. Writes given number of bytes from the given buffer to the byte stream. |

### ByteStream() Member Function

Default constructor.

### Syntax

```
public ByteStream();
```

### ByteStream(System.IO.ByteStream&&) Member Function

Move constructor.

#### Syntax

```
public ByteStream(System.IO.ByteStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.ByteStream&& | |

### operator=(System.IO.ByteStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.ByteStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.ByteStream&& | |

### CopyTo(System.IO.ByteStream&) Member Function

Copies this byte stream to another. Uses default buffer size 16K for copying.

#### Syntax

```
public void CopyTo(System.IO.ByteStream& destination);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| destination | System.IO.ByteStream& | A byte stream to copy to. |

### CopyTo(System.IO.ByteStream&, int) Member Function

Copies this byte stream to another using given buffer size.

#### Syntax

```
public void CopyTo(System.IO.ByteStream& destination, int bufferSize);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| destination | System.IO.ByteStream& | A byte stream to copy to. |
| bufferSize | int | Buffer size. |

### Read(byte*, int) Member Function

Abstract member function that derived byte stream class overrides. Reads at most given number of bytes from the byte stream to the given buffer. Returns number of bytes read that might be less than the number of bytes requested.

#### Syntax

```
public abstract int Read(byte* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | byte* | A buffer to read data to. |
| count | int | Maximum number of bytes to read. |

#### Returns

int

Returns number of bytes read. Might be less than the number of bytes requested but always non-negative. Return value 0 indicates end of stream.

#### Remarks

Throws an exception if reading fails.

### ReadByte() Member Function

Abstract member function that derived byte stream class overrides. Reads one byte from the byte stream. Returns the byte read or -1 if end of stream is encountered.

#### Syntax

```
public abstract int ReadByte();
```

#### Returns

int

Returns the byte read or -1 if end of stream is encountered.

#### Remarks

Throws an exception if reading fails.

### Write(byte) Member Function

Abstract member function that derived byte stream class overrides. Writes one byte of data to the byte stream.

### Syntax

```
public abstract void Write(byte x);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| x | byte | A byte to write. |

### Remarks

Throws an exception if writing fails.

### Write(byte*, int) Member Function

Abstract member function that derived byte stream class overrides. Writes given number of bytes from the given buffer to the byte stream.

#### Syntax

```
public abstract void Write(byte* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | byte* | A buffer of data to write. |
| count | int | Number of bytes to write. |

## 5.10.3 CloseFileException Class

An exception thrown when closing a file fails.

### Syntax

```
public class CloseFileException;
```

### Base Class

System.Exception

#### 5.10.3.1 Member Functions

| Member Function | Description |
|---|---|
| CloseFileException() | Default constructor. |
| CloseFileException(const System.IO.-CloseFileException&) | Copy constructor. |
| operator=(const System.IO.-CloseFileException&) | Copy assignment. |
| CloseFileException(System.IO.-CloseFileException&&) | Move constructor. |
| operator=(System.IO.CloseFileException&&) | Move assignment. |
| ~CloseFileException() | Destructor. |

### CloseFileException() Member Function

Default constructor.

#### Syntax

```
public CloseFileException();
```

### CloseFileException(const System.IO.CloseFileException&) Member Function

Copy constructor.

#### Syntax

```
public CloseFileException(const System.IO.CloseFileException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.CloseFileException& | Argument to copy. |

### operator=(const System.IO.CloseFileException&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.IO.CloseFileException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.CloseFileException& | Argument to assign. |

### CloseFileException(System.IO.CloseFileException&&) Member Function

Move constructor.

#### Syntax

```
public CloseFileException(System.IO.CloseFileException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.CloseFileException&& | Argument to move from. |

### operator=(System.IO.CloseFileException&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.CloseFileException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.CloseFileException&& | Argument to assign from. |

### ~CloseFileException() Member Function

Destructor.

#### Syntax

```
public ~CloseFileException();
```

## 5.10.4 FileByteStream Class

Represents a stream of bytes connected to a file. The stream can be read from or written to but not both.

### Syntax

```
public class FileByteStream;
```

### Base Class

System.IO.ByteStream

#### 5.10.4.1 Member Functions

| Member Function | Description |
|---|---|
| FileByteStream(System.IO.FileByteStream&&) | Move constructor. |
| operator=(System.IO.FileByteStream&&) | Move assignment. |
| Read(byte*, int) | Reads at most given number of bytes from the file to the given buffer. Returns number of bytes |
| ReadByte() | Reads a byte from the file. Returns the byte read, or -1 if end of file is encountered. |
| Write(byte) | Writes a byte of data to the file. |
| Write(byte*, int) | Writes given number of bytes from the given buffer to the file. |
| ~FileByteStream() | Destructor. Closes the file. |

### FileByteStream(System.IO.FileByteStream&&) Member Function

Move constructor.

#### Syntax

```
public FileByteStream(System.IO.FileByteStream&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.FileByteStream&& | A file byte stream to move from. |

### operator=(System.IO.FileByteStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.FileByteStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.FileByteStream&& | A file byte stream to assign from. |

### Read(byte*, int) Member Function

Reads at most given number of bytes from the file to the given buffer. Returns number of bytes

#### Syntax

```
public int Read(byte* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | byte* | A buffer to read to. |
| count | int | Maximum number of bytes to read. |

#### Returns

int

Returns the number of bytes read. Return value of 0 indicates end of file.

#### Remarks

Throws IOException if reading fails.

### ReadByte() Member Function

Reads a byte from the file. Returns the byte read, or -1 if end of file is encountered.

### Syntax

```
public int ReadByte();
```

### Returns

int

Returns the byte read, or -1 if end of file is encountered.

### Remarks

Throws IOException if reading fails.

### Write(byte) Member Function

Writes a byte of data to the file.

### Syntax

```
public void Write(byte x);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| x | byte | A byte to write. |

### Remarks

Throws IOException if writing fails.

### Write(byte*, int) Member Function

Writes given number of bytes from the given buffer to the file.

#### Syntax

```
public void Write(byte* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | byte* | A buffer of data to write. |
| count | int | The number of bytes to write. |

#### Remarks

Throws IOException if writing fails.

### ~FileByteStream() Member Function

Destructor. Closes the file.

#### Syntax

```
public ~FileByteStream();
```

## 5.10.5 FileMapping Class

Represents memory mapped input file.

### Syntax

```
public class FileMapping;
```

### 5.10.5.1 Member Functions

| Member Function | Description |
|---|---|
| FileMapping(System.IO.FileMapping&&) | Move construcor. |
| operator=(System.IO.FileMapping&&) | Move assignment. |
| Begin() const | Returns a pointer to the beginning of the file's data in memory. |
| End() const | Returns a pointer one past the end of the file's data in memory. |
| FilePath() const | Returns the name of the file. |
| ~FileMapping() | Destructor. Closes the file and releases memory. |

### FileMapping(System.IO.FileMapping&&) Member Function

Move construcor.

#### Syntax

```
public FileMapping(System.IO.FileMapping&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.FileMapping&& | A file mapping to move from. |

### operator=(System.IO.FileMapping&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.FileMapping&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.FileMapping&& | A file mapping to assign. |

### Begin() const Member Function

Returns a pointer to the beginning of the file's data in memory.

### Syntax

```
public const char* Begin() const;
```

### Returns

const char*

Returns a pointer to the beginning of the file's data in memory.

### End() const Member Function

Returns a pointer one past the end of the file's data in memory.

#### Syntax

```
public const char* End() const;
```

#### Returns

const char*

Returns a pointer one past the end of the file's data in memory.

### FilePath() const Member Function

Returns the name of the file.

#### Syntax

```
public const System.String<char>& FilePath() const;
```

#### Returns

const System.String<char>&

Returns the name of the file.

### ~FileMapping() Member Function

Destructor. Closes the file and releases memory.

#### Syntax

```
public ~FileMapping();
```

## 5.10.6 FileMappingFailure Class

Exception thrown when opening a memory mapped file fails.

### Syntax

```
public class FileMappingFailure;
```

### Base Class

System.Exception

#### 5.10.6.1 Member Functions

| Member Function | Description |
|---|---|
| FileMappingFailure() | Default constructor. |
| FileMappingFailure(const System.IO.-FileMappingFailure&) | Copy constructor. |
| operator=(const System.IO.-FileMappingFailure&) | Copy assignment. |
| FileMappingFailure(System.IO.-FileMappingFailure&&) | Move constructor. |
| operator=(System.IO.FileMappingFailure&&) | Move assignment. |

### FileMappingFailure() Member Function

Default constructor.

#### Syntax

```
public FileMappingFailure();
```

### FileMappingFailure(const System.IO.FileMappingFailure&) Member Function

Copy constructor.

#### Syntax

```
public FileMappingFailure(const System.IO.FileMappingFailure& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.FileMappingFailure& | Argument to copy. |

### operator=(const System.IO.FileMappingFailure&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.IO.FileMappingFailure& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.FileMappingFailure& | Argument to assign. |

### FileMappingFailure(System.IO.FileMappingFailure&&) Member Function

Move constructor.

#### Syntax

```
public FileMappingFailure(System.IO.FileMappingFailure&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.FileMappingFailure&& | Argument to move from. |

### operator=(System.IO.FileMappingFailure&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.FileMappingFailure&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.FileMappingFailure&& | Argument to assign from. |

## 5.10.7 IOBuffer Class

A handle to dynamically allocated memory.

### Syntax

```
public class IOBuffer;
```

#### 5.10.7.1 Member Functions

| Member Function | Description |
|---|---|
| IOBuffer(System.IO.IOBuffer&&) | Move constructor. |
| operator=(System.IO.IOBuffer&&) | Move assignment. |
| IOBuffer(ulong) | Constructor. Creates an I/O buffer of size_ bytes. |
| Mem() const | Returns a generic pointer to the allocated memory. |
| Size() const | Returns the size of the I/O buffer. |
| ~IOBuffer() | Destructor. Releases the allocated memory back to system. |

### IOBuffer(System.IO.IOBuffer&&) Member Function

Move constructor.

#### Syntax

```
public IOBuffer(System.IO.IOBuffer&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.IOBuffer&& | An I/O buffer to move from. |

### operator=(System.IO.IOBuffer&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.IOBuffer&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.IOBuffer&& | An I/O buffer to assign from. |

### IOBuffer(ulong) Member Function

Constructor. Creates an I/O buffer of size_ bytes.

#### Syntax

```
public IOBuffer(ulong size_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| size_ | ulong | Number of bytes to allocate. |

### Mem() const Member Function

Returns a generic pointer to the allocated memory.

#### Syntax

```
public void* Mem() const;
```

#### Returns

void*

Returns a generic pointer to the allocated memory.

### Size() const Member Function

Returns the size of the I/O buffer.

### Syntax

```
public ulong Size() const;
```

### Returns

ulong

Returns the size of the I/O buffer.

### ~IOBuffer() Member Function

Destructor. Releases the allocated memory back to system.

#### Syntax

```
public ~IOBuffer();
```

## 5.10.8 IOException Class

An exception thrown when an I/O operation fails.

### Syntax

```
public class IOException;
```

### Base Class

System.Exception

#### 5.10.8.1 Member Functions

| Member Function | Description |
|---|---|
| IOException() | Default constructor. |
| IOException(const System.IO.IOException&) | Copy constructor. |
| operator=(const System.IO.IOException&) | Copy assignment. |
| IOException(System.IO.IOException&&) | Move constructor. |
| operator=(System.IO.IOException&&) | Move assignment. |
| ~IOException() | Destructor. |

### IOException() Member Function

Default constructor.

### Syntax

```
public IOException();
```

### IOException(const System.IO.IOException&) Member Function

Copy constructor.

#### Syntax

```
public IOException(const System.IO.IOException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.IOException& | Argument to copy. |

### operator=(const System.IO.IOException&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.IO.IOException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.IOException& | Argument to assign. |

### IOException(System.IO.IOException&&) Member Function

Move constructor.

#### Syntax

```
public IOException(System.IO.IOException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.IOException&& | Argument to move from. |

### operator=(System.IO.IOException&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.IOException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.IOException&& | Argument to assign from. |

### ~IOException() Member Function

Destructor.

#### Syntax

```
public ~IOException();
```

## 5.10.9 InputFileStream Class

A stream of characters connected to an input file.

### Syntax

```
public class InputFileStream;
```

### Base Class

System.IO.InputStream

#### 5.10.9.1 Member Functions

| Member Function | Description |
|---|---|
| InputFileStream() | Default constructor. Initializes the input file stream and connects it to standard input stream. |
| InputFileStream(System.IO.InputFileStream&&) | Move constructor. |
| operator=(System.IO.InputFileStream&&) | Move assignment. |
| Close() | If the input file stream is connected to an open file, closes the file, otherwise throws CloseFileException; |
| EndOfStream() const | Returns true if the end of the stream is encountered, false otherwise. |
| FileName() const | Returns the name of the file input file stream is connected to. |
| Handle() const | Returns the file handler the input file stream is connected to. |
| InputFileStream(int, uint) | Constructor. Initializes the input file stream with the given buffer size and connects it to the file with the given file handle. |
| ReadLine() | Reads a line of text from the input file stream and returns it. |
| ReadToEnd() | Reads the content of the input file into a string and returns it. |
| ~InputFileStream() | Destructor. If the input file stream is connected to an open file, closes the file. |

### InputFileStream() Member Function

Default constructor. Initializes the input file stream and connects it to standard input stream.

#### Syntax

```
public InputFileStream();
```

### InputFileStream(System.IO.InputFileStream&&) Member Function

Move constructor.

#### Syntax

```
public InputFileStream(System.IO.InputFileStream&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.InputFileStream&& | An input file stream to move from. |

### operator=(System.IO.InputFileStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.InputFileStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.InputFileStream&& | An input file steram to assign from. |

### Close() Member Function

If the input file stream is connected to an open file, closes the file, otherwise throws CloseFileException;

### Syntax

```
public void Close();
```

### EndOfStream() const Member Function

Returns true if the end of the stream is encountered, false otherwise.

#### Syntax

```
public bool EndOfStream() const;
```

#### Returns

bool

Returns true if the end of the stream is encountered, false otherwise.

### FileName() const Member Function

Returns the name of the file input file stream is connected to.

#### Syntax

```
public const System.String<char>& FileName() const;
```

#### Returns

const System.String<char>&

Returns the name of the file input file stream is connected to.

### Handle() const Member Function

Returns the file handler the input file stream is connected to.

#### Syntax

```
public int Handle() const;
```

#### Returns

int

Returns the file handler the input file stream is connected to.

### InputFileStream(int, uint) Member Function

Constructor. Initializes the input file stream with the given buffer size and connects it to the file with the given file handle.

#### Syntax

```
public InputFileStream(int handle_, uint bufferSize_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| handle_ | int | A file handle. |
| bufferSize_ | uint | The size of the input buffer. |

### ReadLine() Member Function

Reads a line of text from the input file stream and returns it.

#### Syntax

```
public System.String<char> ReadLine();
```

#### Returns

System.String<char>

Returns the line of text read.

### ReadToEnd() Member Function

Reads the content of the input file into a string and returns it.

### Syntax

```
public System.String<char> ReadToEnd();
```

### Returns

System.String<char>

Returns the contents of the file read.

### ~InputFileStream() Member Function

Destructor. If the input file stream is connected to an open file, closes the file.

#### Syntax

```
public ~InputFileStream();
```

## 5.10.10 InputStream Class

An abstract base class for input stream classes.

### Syntax

```
public abstract class InputStream;
```

#### 5.10.10.1 Member Functions

| Member Function | Description |
|---|---|
| InputStream() | Default constructor. |
| InputStream(System.IO.InputStream&&) | Move constructor. |
| operator=(System.IO.InputStream&&) | Move assignment. |
| EndOfStream() const | Abstract member function for returning the end-of-file status. |
| ReadLine() | Abstract member function for reading a line of text from the input stream. |
| ReadToEnd() | Abstract member function for reading the contents of the stream into a string. |
| ~InputStream() | Destructor. |

### InputStream() Member Function

Default constructor.

#### Syntax

```
public InputStream();
```

### InputStream(System.IO.InputStream&&) Member Function

Move constructor.

#### Syntax

```
public InputStream(System.IO.InputStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.InputStream&& | |

### operator=(System.IO.InputStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.InputStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.InputStream&& | |

### EndOfStream() const Member Function

Abstract member function for returning the end-of-file status.

#### Syntax

```
public abstract bool EndOfStream() const;
```

#### Returns

bool

Returns true if the end of the stream is encountered, false otherwise.

### ReadLine() Member Function

Abstract member function for reading a line of text from the input stream.

#### Syntax

```
public abstract System.String<char> ReadLine();
```

#### Returns

System.String<char>

Returns the line read.

### ReadToEnd() Member Function

Abstract member function for reading the contents of the stream into a string.

#### Syntax

```
public abstract System.String<char> ReadToEnd();
```

#### Returns

System.String<char>

Returns the contents of the stream.

### ~InputStream() Member Function

Destructor.

#### Syntax

```
public ~InputStream();
```

## 5.10.11 InputStringStream Class

A class for reading from a string.

### Syntax

```
public class InputStringStream;
```

### Base Class

System.IO.InputStream

#### 5.10.11.1 Member Functions

| Member Function | Description |
|---|---|
| InputStringStream() | Default constructor. |
| InputStringStream(System.IO.-InputStringStream&&) | Move constructor. |
| operator=(System.IO.InputStringStream&&) | Move assignment. |
| EndOfStream() const | Returns true if the end of the string has been encountered, false otherwise. |
| GetStr() const | Returns the contained string. |
| InputStringStream(const System.String<char>-&) | Constructor. Initializes the input string stream with the given string. |
| ReadLine() | Reads a line of text from the string and returns it. |
| ReadToEnd() | Returns the contents of the rest of the string. |
| SetStr(const System.String<char>&) | Sets the contained string. |
| ~InputStringStream() | Destructor. |

### InputStringStream() Member Function

Default constructor.

#### Syntax

```
public InputStringStream();
```

### InputStringStream(System.IO.InputStringStream&&) Member Function

Move constructor.

#### Syntax

```
public InputStringStream(System.IO.InputStringStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.InputStringStream&& | |

### operator=(System.IO.InputStringStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.InputStringStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.InputStringStream&& | |

### EndOfStream() const Member Function

Returns true if the end of the string has been encountered, false otherwise.

#### Syntax

```
public bool EndOfStream() const;
```

#### Returns

bool

Returns true if the end of the string has been encountered, false otherwise.

### GetStr() const Member Function

Returns the contained string.

#### Syntax

```
public const System.String<char>& GetStr() const;
```

#### Returns

const System.String<char>&

Returns the contained string.

### InputStringStream(const System.String<char>&) Member Function

Constructor. Initializes the input string stream with the given string.

#### Syntax

```
public InputStringStream(const System.String<char>& str_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| str_ | const System.String<char>& | A string to read from. |

### ReadLine() Member Function

Reads a line of text from the string and returns it.

#### Syntax

```
public System.String<char> ReadLine();
```

#### Returns

System.String<char>

Returns the line read.

### ReadToEnd() Member Function

Returns the contents of the rest of the string.

#### Syntax

```
public System.String<char> ReadToEnd();
```

#### Returns

System.String<char>

Returns the contents of the rest of the string.

### SetStr(const System.String<char>&) Member Function

Sets the contained string.

#### Syntax

```
public void SetStr(const System.String<char>& str_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| str_ | const System.String<char>& | A string. |

### ~InputStringStream() Member Function

Destructor.

#### Syntax

```
public ~InputStringStream();
```

## 5.10.12 InvalidPathException Class

An exception class that is thrown if a path contains too many .. components.

### Syntax

```
public class InvalidPathException;
```

### Base Class

System.Exception

#### 5.10.12.1 Member Functions

| Member Function | Description |
|---|---|
| InvalidPathException() | Default constructor. |
| InvalidPathException(const System.IO.-InvalidPathException&) | Copy constructor. |
| operator=(const System.IO.-InvalidPathException&) | Copy assignment. |
| InvalidPathException(System.IO.-InvalidPathException&&) | Move constructor. |
| operator=(System.IO.InvalidPathException&&) | Move assignment. |
| InvalidPathException(const System.String<-char>&) | Constructor. Initializes the exception with the given error message. |
| ~InvalidPathException() | Destructor. |

### InvalidPathException() Member Function

Default constructor.

#### Syntax

```
public InvalidPathException();
```

### InvalidPathException(const System.IO.InvalidPathException&) Member Function

Copy constructor.

#### Syntax

```
public InvalidPathException(const System.IO.InvalidPathException& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | const System.IO.InvalidPathException& | |

### operator=(const System.IO.InvalidPathException&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.IO.InvalidPathException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.InvalidPathException& | Argument to assign. |

### InvalidPathException(System.IO.InvalidPathException&&) Member Function

Move constructor.

#### Syntax

```
public InvalidPathException(System.IO.InvalidPathException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.InvalidPathException&& | Argument to move from. |

### operator=(System.IO.InvalidPathException&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.InvalidPathException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.InvalidPathException&& | Argument to assign from. |

### InvalidPathException(const System.String<char>&) Member Function

Constructor. Initializes the exception with the given error message.

#### Syntax

```
public InvalidPathException(const System.String<char>& message_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| message_ | const System.String<char>& | An error message. |

### ~InvalidPathException() Member Function

Destructor.

#### Syntax

```
public ~InvalidPathException();
```

## 5.10.13 MemoryByteStream Class

Represents a stream of bytes in memory. Writing to the stream writes data to the end of the stream. Reading from the stream reads data starting from the beginning of the stream.

### Syntax

```
public class MemoryByteStream;
```

### Base Class

System.IO.ByteStream

### 5.10.13.1 Member Functions

| Member Function | Description |
|---|---|
| MemoryByteStream() | Default constructor. Constructs an empty memory byte strewam. |
| MemoryByteStream(System.IO.-MemoryByteStream&&) | Move constructor. |
| operator=(System.IO.MemoryByteStream&&) | Move assignment. |
| Data() const | Returns a pointer to the beginning of data. |
| MemoryByteStream(byte*, int) | Constructor. Initializes the memory byte stream with given data. |
| Read(byte*, int) | Reads at most given number of bytes from the memory stream to the given buffer. Returns the number of bytes read. Return value of 0 indicates end of stream. |
| ReadByte() | Reads one byte of data from the stream. Returns the byte read, or -1 if end of stream is encountered. |
| ReadPos() const | Returns reading position. |
| Size() const | Returns the number of bytes contained in the stream. |
| Write(byte) | Writes one byte of data to the end of the stream. |
| Write(byte*, int) | Writes given number of bytes from the given buffer to the end of the stream. |
| ~MemoryByteStream() | Destructor. Releases memory occupied by the stream. |

### MemoryByteStream() Member Function

Default constructor. Constructs an empty memory byte strewam.

#### Syntax

```
public MemoryByteStream();
```

### MemoryByteStream(System.IO.MemoryByteStream&&) Member Function

Move constructor.

#### Syntax

```
public MemoryByteStream(System.IO.MemoryByteStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.MemoryByteStream&& | |

### operator=(System.IO.MemoryByteStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.MemoryByteStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.MemoryByteStream&& | |

### Data() const Member Function

Returns a pointer to the beginning of data.

#### Syntax

```
public const byte* Data() const;
```

#### Returns

const byte*

Returns a pointer to the beginning of data.

### MemoryByteStream(byte*, int) Member Function

Constructor. Initializes the memory byte stream with given data.

#### Syntax

```
public MemoryByteStream(byte* data, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| data | byte* | Buffer of data. |
| count | int | Number of bytes in data buffer. |

### Read(byte*, int) Member Function

Reads at most given number of bytes from the memory stream to the given buffer. Returns the number of bytes read. Return value of 0 indicates end of stream.

#### Syntax

```
public int Read(byte* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | byte* | A buffer to read to. |
| count | int | Maximum number of bytes to read. |

#### Returns

int

Returns the number of bytes read.

### ReadByte() Member Function

Reads one byte of data from the stream. Returns the byte read, or -1 if end of stream is encountered.

#### Syntax

```
public int ReadByte();
```

#### Returns

int

Returns the byte read, or -1 if end of stream is encountered.

### ReadPos() const Member Function

Returns reading position.

#### Syntax

```
public int ReadPos() const;
```

#### Returns

int

Returns reading position.

### Size() const Member Function

Returns the number of bytes contained in the stream.

### Syntax

```
public int Size() const;
```

### Returns

int

Returns the number of bytes contained in the stream.

### Write(byte) Member Function

Writes one byte of data to the end of the stream.

### Syntax

```
public void Write(byte x);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| x | byte | A byte to write. |

### Write(byte*, int) Member Function

Writes given number of bytes from the given buffer to the end of the stream.

#### Syntax

```
public void Write(byte* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | byte* | A buffer of data to write. |
| count | int | Number of bytes to write. |

### ~MemoryByteStream() Member Function

Destructor. Releases memory occupied by the stream.

#### Syntax

```
public ~MemoryByteStream();
```

## 5.10.14 OpenFileException Class

An exception class thrown if opening a file fails.

### Syntax

```
public class OpenFileException;
```

### Base Class

System.Exception

#### 5.10.14.1 Member Functions

| Member Function | Description |
|---|---|
| OpenFileException() | Default constructor. |
| OpenFileException(const System.IO.-OpenFileException&) | Copy constructor. |
| operator=(const System.IO.-OpenFileException&) | Copy assignment. |
| OpenFileException(System.IO.-OpenFileException&&) | Move constructor. |
| operator=(System.IO.OpenFileException&&) | Move assignment. |
| OpenFileException(const System.String<char>-&) | Constructor. Initializes the exception with the given error message. |
| ~OpenFileException() | Destructor. |

### OpenFileException() Member Function

Default constructor.

#### Syntax

```
public OpenFileException();
```

### OpenFileException(const System.IO.OpenFileException&) Member Function

Copy constructor.

#### Syntax

```
public OpenFileException(const System.IO.OpenFileException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.OpenFileException& | Argument to copy. |

### operator=(const System.IO.OpenFileException&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.IO.OpenFileException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.IO.OpenFileException& | Argument to assign. |

### OpenFileException(System.IO.OpenFileException&&) Member Function

Move constructor.

#### Syntax

```
public OpenFileException(System.IO.OpenFileException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.OpenFileException&& | Argument to move from. |

### operator=(System.IO.OpenFileException&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.OpenFileException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.OpenFileException&& | Argument to assign from. |

### OpenFileException(const System.String<char>&) Member Function

Constructor. Initializes the exception with the given error message.

#### Syntax

```
public OpenFileException(const System.String<char>& message_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| message_ | const System.String<char>& | An error message. |

### ~OpenFileException() Member Function

Destructor.

#### Syntax

```
public ~OpenFileException();
```

## 5.10.15 OutputFileStream Class

A stream of characters connected to an output file.

### Syntax

```
public class OutputFileStream;
```

### Base Class

System.IO.OutputStream

#### 5.10.15.1 Member Functions

| Member Function | Description |
|---|---|
| OutputFileStream() | Default constructor. Connects the output file stream with standard output stream. |
| OutputFileStream(System.IO.-OutputFileStream&&) | Move constructor. |
| operator=(System.IO.OutputFileStream&&) | Move assignment. |
| Close() | If the output file stream is connected to an open file, the file is closed, otherwise throws CloseFile-Exception. |
| FileName() const | Returns the name of the output file the output file stream is connected to. |
| Handle() const | Returns the file handle of the file the output file stream is connected to. |
| Open(const System.String<char>&) | Opens the given output file and connects it to the output file stream. |
| Open(const System.String<char>&, bool) | Opens the given output file and connects it to the output file stream. |
| Open(const System.String<char>&, int) | Opens the given output file and connects it to the output file stream using given permissions. |
| Open(const System.String<char>&, int, bool) | Opens the given output file and connects it to the output file stream using given permissions. |
| OutputFileStream(const System.String<char>&-) | Constructor. Connects the output file stream with the output file of the given name. |
| OutputFileStream(const System.String<char>&, bool) | Constructor. Connects the output file stream to an output file of the given name. |

| | |
|---|---|
| OutputFileStream(const System.String<char>&, int) | Constructor. Connects the output file stream to an output file with the given name using given permissions. |
| OutputFileStream(const System.String<char>&, int, bool) | Constructor. Connects the output file stream to an output file with the given name using given permissions. |
| OutputFileStream(int) | Constructor. Connects the output file stream to an output file with the given file handle. |
| Write(bool) | Writes the given Boolean value to the output file stream. |
| Write(byte) | Writes the given **byte** to the output file stream. |
| Write(char) | Writes the given character to the output file stream. |
| Write(const System.String<char>&) | Writes the given string to the output file stream. |
| Write(const System.String<uchar>&) | Writes the given UTF-32 string to the output file stream. |
| Write(const System.String<wchar>&) | Writes the given UTF-16 string to the output file stream. |
| Write(const char*) | Writes the given C-style string to the output file stream. |
| Write(double) | Writes the given **double** value to the output file stream. |
| Write(float) | Writes the given **float** value to the output file stream. |
| Write(int) | Writes the given **int** value to the output file stream. |
| Write(long) | Writes the given **long** value to the output file stream. |
| Write(sbyte) | Writes the given **sbyte** value to the output file stream. |
| Write(short) | Writes the given **short** value to the output file stream. |

| | |
|---|---|
| Write(uchar) | Writes the given **uchar** value to the output file stream as a sequence if UTF-8 encoded bytes. |
| Write(uint) | Writes the given **uint** value to the output file stream. |
| Write(ulong) | Writes the given **ulong** value to the output file stream. |
| Write(ushort) | Writes the given **ushort** value to the output file stream. |
| Write(wchar) | Writes the given **wchar** value to the output file stream as a sequence if UTF-8 encoded bytes. |
| WriteLine() | Writes a new line to the output file stream. |
| WriteLine(bool) | Writes the given Boolean value followed by a new line to the output file stream. |
| WriteLine(byte) | Writes the given **byte** value followed by a new line to the output file stream. |
| WriteLine(char) | Writes the given character followed by a newline to the output file stream. |
| WriteLine(const System.String<char>&) | Writes the given string followed by a newline to the output file stream. |
| WriteLine(const System.String<uchar>&) | Writes the given UTF-32 string followed by a newline to the output file stream. |
| WriteLine(const System.String<wchar>&) | Writes the given UTF-16 string followed by a newline to the output file stream. |
| WriteLine(const char*) | Writes the given C-style string followed by a new line to the output file stream. |
| WriteLine(double) | Writes the given **double** value followed by a new line to the output file stream. |
| WriteLine(float) | Writes the given **float** value followed by a new line to the output file stream. |
| WriteLine(int) | Writes the given **int** value followed by a new line to the output file stream. |

| | |
|---|---|
| WriteLine(long) | Writes the given **long** value followed by a new line to the output file stream. |
| WriteLine(sbyte) | Writes the given **sbyte** value followed by a new line to the output file stream. |
| WriteLine(short) | Writes the given **short** value followed by a new line to the output file stream. |
| WriteLine(uchar) | Writes the given **uchar** value followed by a newline to the output file stream. The value is written as a sequence of UTF-8 encoded bytes. |
| WriteLine(uint) | Writes the given **uint** value followed by a new line to the output file stream. |
| WriteLine(ulong) | Writes the given **ulong** value followed by a new line to the output file stream. |
| WriteLine(ushort) | Writes the given **ushort** value followed by a new line to the output file stream. |
| WriteLine(wchar) | Writes the given **wchar** value followed by a newline to the output file stream. The value is written as a sequence of UTF-8 encoded bytes. |
| ~OutputFileStream() | Destructor. If the output file stream is connected to an open file, the file is closed. |

### OutputFileStream() Member Function

Default constructor. Connects the output file stream with standard output stream.

#### Syntax

```
public OutputFileStream();
```

### OutputFileStream(System.IO.OutputFileStream&&) Member Function

Move constructor.

#### Syntax

```
public OutputFileStream(System.IO.OutputFileStream&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.IO.OutputFileStream&& | An output file stream to move from. |

### operator=(System.IO.OutputFileStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.OutputFileStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.OutputFileStream&& | |

### Close() Member Function

If the output file stream is connected to an open file, the file is closed, otherwise throws CloseFileException.

### Syntax

```
public void Close();
```

### FileName() const Member Function

Returns the name of the output file the output file stream is connected to.

#### Syntax

```
public const System.String<char>& FileName() const;
```

#### Returns

const System.String<char>&

Returns the name of the output file.

### Handle() const Member Function

Returns the file handle of the file the output file stream is connected to.

#### Syntax

```
public int Handle() const;
```

#### Returns

int

Returns the file handle.

### Open(const System.String<char>&) Member Function

Opens the given output file and connects it to the output file stream.

#### Syntax

```
public void Open(const System.String<char>& fileName_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName_ | const System.String<char>& | The name of the output file. |

#### Remarks

If the output file stream was connected to an open file before the operation, that file is closed first. If the given file does not exist, it is created. If the given file exists, it is truncated to zero length.

### Open(const System.String<char>&, bool) Member Function

Opens the given output file and connects it to the output file stream.

#### Syntax

```
public void Open(const System.String<char>& fileName_, bool append);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName_ | const System.String<char>& | The name of the output file. |
| append | bool | If true, the file is opened for appending. |

#### Remarks

If the append parameter is true, opens the file for appending.

### Open(const System.String<char>&, int) Member Function

Opens the given output file and connects it to the output file stream using given permissions.

#### Syntax

```
public void Open(const System.String<char>& fileName_, int pmode);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName_ | const System.String<char>& | The name of the output file. |
| pmode | int | A permission mode that is formed by ORing together following constants: S_IREAD, S_IWRITE (Windows); S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH (Unix). |

### Open(const System.String<char>&, int, bool) Member Function

Opens the given output file and connects it to the output file stream using given permissions.

#### Syntax

```
public void Open(const System.String<char>& fileName_, int pmode, bool append);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName_ | const System.String<char>& | The name of the output file. |
| pmode | int | A permission mode that is formed by ORing together following constants: S_IREAD, S_IWRITE (Windows); S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH (Unix). |
| append | bool | If true, the file is opened for appending. |

#### Remarks

If the append parameter is true, opens the file for appending.

### OutputFileStream(const System.String<char>&) Member Function

Constructor. Connects the output file stream with the output file of the given name.

#### Syntax

```
public OutputFileStream(const System.String<char>& fileName_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName_ | const System.String<char>& | The name of the output file. |

#### Remarks

If the file does not exist, it is created. If the file exists, it is truncated to zero length.

### OutputFileStream(const System.String<char>&, bool) Member Function

Constructor. Connects the output file stream to an output file of the given name.

#### Syntax

```
public OutputFileStream(const System.String<char>& fileName_, bool append);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName_ | const System.String<char>& | The name of the output file. |
| append | bool | If true, the file is opened for appending. |

#### Remarks

If the append parameter is true, opens the file for appending.

### OutputFileStream(const System.String<char>&, int) Member Function

Constructor. Connects the output file stream to an output file with the given name using given permissions.

#### Syntax

```
public OutputFileStream(const System.String<char>& fileName_, int pmode);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName_ | const System.String<char>& | The name of the output file. |
| pmode | int | A permission mode that is formed by ORing together following constants: S_IREAD, S_IWRITE (Windows); S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH (Unix). |

### OutputFileStream(const System.String<char>&, int, bool) Member Function

Constructor. Connects the output file stream to an output file with the given name using given permissions.

#### Syntax

```
public OutputFileStream(const System.String<char>& fileName_, int pmode, bool append);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName_ | const System.String<char>& | The name of the output file. |
| pmode | int | A permission mode that is formed by ORing together following constants: S_IREAD, S_IWRITE (Windows); S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH (Unix). |
| append | bool | If true, the file is opened for appending. |

#### Remarks

If the append parameter is true, opens the file for appending.

### OutputFileStream(int) Member Function

Constructor. Connects the output file stream to an output file with the given file handle.

#### Syntax

```
public OutputFileStream(int handle_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| handle_ | int | A file handle. |

### Write(bool) Member Function

Writes the given Boolean value to the output file stream.

#### Syntax

```
public void Write(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean value to write. |

#### Remarks

If the value is true, writes “true”, otherwise writes “false”.

### Write(byte) Member Function

Writes the given **byte** to the output file stream.

#### Syntax

```
public void Write(byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A **byte** to write. |

### Write(char) Member Function

Writes the given character to the output file stream.

#### Syntax

```
public void Write(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### Write(const System.String<char>&) Member Function

Writes the given string to the output file stream.

#### Syntax

```
public void Write(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to write. |

### Write(const System.String<uchar>&) Member Function

Writes the given UTF-32 string to the output file stream.

#### Syntax

```
public void Write(const System.String<uchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to write. |

### Write(const System.String<wchar>&) Member Function

Writes the given UTF-16 string to the output file stream.

#### Syntax

```
public void Write(const System.String<wchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to write. |

### Write(const char*) Member Function

Writes the given C-style string to the output file stream.

#### Syntax

```
public void Write(const char* s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### Write(double) Member Function

Writes the given **double** value to the output file stream.

### Syntax

```
public void Write(double d);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A **double** to write. |

### Write(float) Member Function

Writes the given **float** value to the output file stream.

#### Syntax

```
public void Write(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A **float** to write. |

### Write(int) Member Function

Writes the given **int** value to the output file stream.

### Syntax

```
public void Write(int i);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An **int** to write. |

### Write(long) Member Function

Writes the given **long** value to the output file stream.

#### Syntax

```
public void Write(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A **long** to write. |

### Write(sbyte) Member Function

Writes the given **sbyte** value to the output file stream.

### Syntax

```
public void Write(sbyte s);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | sbyte | An **sbyte** to write. |

### Write(short) Member Function

Writes the given **short** value to the output file stream.

#### Syntax

```
public void Write(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A **short** to write. |

### Write(uchar) Member Function

Writes the given **uchar** value to the output file stream as a sequence if UTF-8 encoded bytes.

#### Syntax

```
public void Write(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A **uchar** to write. |

### Write(uint) Member Function

Writes the given **uint** value to the output file stream.

#### Syntax

```
public void Write(uint i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | uint | A **uint** to write. |

### Write(ulong) Member Function

Writes the given **ulong** value to the output file stream.

#### Syntax

```
public void Write(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | A **ulong** to write. |

### Write(ushort) Member Function

Writes the given **ushort** value to the output file stream.

#### Syntax

```
public void Write(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | A **ushort** to write. |

### Write(wchar) Member Function

Writes the given **wchar** value to the output file stream as a sequence if UTF-8 encoded bytes.

#### Syntax

```
public void Write(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A **wchar** to write. |

### WriteLine() Member Function

Writes a new line to the output file stream.

### Syntax

```
public void WriteLine();
```

### WriteLine(bool) Member Function

Writes the given Boolean value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean value to write. |

### WriteLine(byte) Member Function

Writes the given **byte** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A **byte** to write. |

### WriteLine(char) Member Function

Writes the given character followed by a newline to the output file stream.

#### Syntax

```
public void WriteLine(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### WriteLine(const System.String<char>&) Member Function

Writes the given string followed by a newline to the output file stream.

#### Syntax

```
public void WriteLine(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to write. |

### WriteLine(const System.String<uchar>&) Member Function

Writes the given UTF-32 string followed by a newline to the output file stream.

#### Syntax

```
public void WriteLine(const System.String<uchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to write. |

### WriteLine(const System.String<wchar>&) Member Function

Writes the given UTF-16 string followed by a newline to the output file stream.

#### Syntax

```
public void WriteLine(const System.String<wchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to write. |

### WriteLine(const char*) Member Function

Writes the given C-style string followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(const char* s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### WriteLine(double) Member Function

Writes the given **double** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A **double** to write. |

### WriteLine(float) Member Function

Writes the given **float** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A **float** to write. |

### WriteLine(int) Member Function

Writes the given **int** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(int i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An **int** to write. |

### WriteLine(long) Member Function

Writes the given **long** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A **long** to write. |

### WriteLine(sbyte) Member Function

Writes the given **sbyte** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(sbyte s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | sbyte | An **sbyte** to write. |

### WriteLine(short) Member Function

Writes the given **short** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A **short** to write. |

### WriteLine(uchar) Member Function

Writes the given **uchar** value followed by a newline to the output file stream. The value is written as a sequence of UTF-8 encoded bytes.

#### Syntax

```
public void WriteLine(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A **uchar** to write. |

### WriteLine(uint) Member Function

Writes the given **uint** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(uint u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | uint | A **uint** to write. |

### WriteLine(ulong) Member Function

Writes the given **ulong** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | A **ulong** to write. |

### WriteLine(ushort) Member Function

Writes the given **ushort** value followed by a new line to the output file stream.

#### Syntax

```
public void WriteLine(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | A **ushort** to write. |

### WriteLine(wchar) Member Function

Writes the given **wchar** value followed by a newline to the output file stream. The value is written as a sequence of UTF-8 encoded bytes.

#### Syntax

```
public void WriteLine(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A **wchar** to write. |

### ~OutputFileStream() Member Function

Destructor. If the output file stream is connected to an open file, the file is closed.

#### Syntax

```
public ~OutputFileStream();
```

## 5.10.16 OutputStream Class

An abstract base class for output stream classes.

### Syntax

```
public abstract class OutputStream;
```

### 5.10.16.1 Member Functions

| Member Function | Description |
|---|---|
| OutputStream() | Default constructor. |
| OutputStream(System.IO.OutputStream&&) | Move constructor. |
| operator=(System.IO.OutputStream&&) | Move assignment. |
| Write(bool) | Writes a Boolean value to output stream. |
| Write(byte) | Writes a **byte** to the output stream. |
| Write(char) | Writes a character to the output stream. |
| Write(const System.String<char>&) | Writes a string to the output stream. |
| Write(const System.String<uchar>&) | Writes a UTF-32 string to the output stream. |
| Write(const System.String<wchar>&) | Writes a UTF-16 string to the output stream. |
| Write(const char*) | Writes a C-style string to the output steram. |
| Write(double) | Writes a **double** to the output stream. |
| Write(float) | Writes a **float** to the output stream. |
| Write(int) | Writes an **int** to the output stream. |
| Write(long) | Writes a **long** to the output stream. |
| Write(sbyte) | Writes an **sbyte** to the output stream. |
| Write(short) | Writes a **short** to the output stream. |
| Write(uchar) | Writes a **uchar** to the output stream. |
| Write(uint) | Writes a **uint** to the output stream. |
| Write(ulong) | Writes a **ulong** to the output stream. |

| | |
|---|---|
| Write(ushort) | Writes a **ushort** to the output stream. |
| Write(wchar) | Writes a **wchar** to the output stream. |
| WriteLine() | Writes a new line to the output stream. |
| WriteLine(bool) | Writes a Boolean followed by a new line to the output stream. |
| WriteLine(byte) | Writes a **byte** followed by a new line to the output stream. |
| WriteLine(char) | Writes a character followed by a new line to the output stream. |
| WriteLine(const System.String<char>&) | Writes a string followed by a newline to the output stream. |
| WriteLine(const System.String<uchar>&) | Writes a UTF-32 string followed by a newline to the output stream. |
| WriteLine(const System.String<wchar>&) | Writes a UTF-16 string followed by a newline to the output stream. |
| WriteLine(const char*) | Writes a C-style string followed by a new line to the output stream. |
| WriteLine(double) | Writes a **double** followed by a new line to the output stream. |
| WriteLine(float) | Writes a **float** followed by a new line to the output stream. |
| WriteLine(int) | Writes an **int** followed by a new line to the output stream. |
| WriteLine(long) | Writes a **long** followed by a new line to the output stream. |
| WriteLine(sbyte) | Writes an **sbyte** followed by a new line to the output stream. |
| WriteLine(short) | Writes a **short** followed by a new line to the output stream. |
| WriteLine(uchar) | Writes a **uchar** followed by a newline to the output stream. |

| | |
|---|---|
| WriteLine(uint) | Writes a **uint** followed by a new line to the output stream. |
| WriteLine(ulong) | Writes a **ulong** followed by a new line to the output stream. |
| WriteLine(ushort) | Writes a **ushort** followed by a new line to the output stream. |
| WriteLine(wchar) | Writes a **wchar** followed by a newline to the output stream. |
| ~OutputStream() | Destructor. |

### OutputStream() Member Function

Default constructor.

#### Syntax

```
public OutputStream();
```

### OutputStream(System.IO.OutputStream&&) Member Function

Move constructor.

#### Syntax

```
public OutputStream(System.IO.OutputStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.OutputStream&& | |

### operator=(System.IO.OutputStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.OutputStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.OutputStream&& | |

### Write(bool) Member Function

Writes a Boolean value to output stream.

#### Syntax

```
public abstract void Write(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean to write. |

### Write(byte) Member Function

Writes a **byte** to the output stream.

#### Syntax

```
public abstract void Write(byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A **byte** to write. |

### Write(char) Member Function

Writes a character to the output stream.

#### Syntax

```
public abstract void Write(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### Write(const System.String<char>&) Member Function

Writes a string to the output stream.

#### Syntax

```
public abstract void Write(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to write. |

### Write(const System.String<uchar>&) Member Function

Writes a UTF-32 string to the output stream.

#### Syntax

```
public abstract void Write(const System.String<uchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to write. |

### Write(const System.String<wchar>&) Member Function

Writes a UTF-16 string to the output stream.

#### Syntax

```
public abstract void Write(const System.String<wchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to write. |

### Write(const char*) Member Function

Writes a C-style string to the output steram.

#### Syntax

```
public abstract void Write(const char* s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### Write(double) Member Function

Writes a **double** to the output stream.

#### Syntax

```
public abstract void Write(double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A **double** to write. |

### Write(float) Member Function

Writes a **float** to the output stream.

#### Syntax

```
public abstract void Write(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A **float** to write. |

### Write(int) Member Function

Writes an **int** to the output stream.

### Syntax

```
public abstract void Write(int i);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An **int** to write. |

### Write(long) Member Function

Writes a **long** to the output stream.

#### Syntax

```
public abstract void Write(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A **long** to write. |

### Write(sbyte) Member Function

Writes an **sbyte** to the output stream.

#### Syntax

```
public abstract void Write(sbyte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | sbyte | An **sbyte** to write. |

### Write(short) Member Function

Writes a **short** to the output stream.

#### Syntax

```
public abstract void Write(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A **short** to write. |

### Write(uchar) Member Function

Writes a **uchar** to the output stream.

### Syntax

```
public abstract void Write(uchar c);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A **uchar** to write. |

### Write(uint) Member Function

Writes a **uint** to the output stream.

#### Syntax

```
public abstract void Write(uint i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | uint | A **uint** to write. |

### Write(ulong) Member Function

Writes a **ulong** to the output stream.

#### Syntax

```
public abstract void Write(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | A **ulong** to write. |

### Write(ushort) Member Function

Writes a **ushort** to the output stream.

#### Syntax

```
public abstract void Write(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | A **ushort** to write. |

### Write(wchar) Member Function

Writes a **wchar** to the output stream.

#### Syntax

```
public abstract void Write(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A **wchar** to write. |

### WriteLine() Member Function

Writes a new line to the output stream.

### Syntax

```
public abstract void WriteLine();
```

### WriteLine(bool) Member Function

Writes a Boolean followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean to write. |

### WriteLine(byte) Member Function

Writes a **byte** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A **byte** to write. |

### WriteLine(char) Member Function

Writes a character followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### WriteLine(const System.String<char>&) Member Function

Writes a string followed by a newline to the output stream.

#### Syntax

```
public abstract void WriteLine(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to write. |

### WriteLine(const System.String<uchar>&) Member Function

Writes a UTF-32 string followed by a newline to the output stream.

#### Syntax

```
public abstract void WriteLine(const System.String<uchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to write. |

### WriteLine(const System.String<wchar>&) Member Function

Writes a UTF-16 string followed by a newline to the output stream.

#### Syntax

```
public abstract void WriteLine(const System.String<wchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to write. |

### WriteLine(const char*) Member Function

Writes a C-style string followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(const char* s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### WriteLine(double) Member Function

Writes a **double** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A **double** to write. |

### WriteLine(float) Member Function

Writes a **float** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A **float** to write. |

### WriteLine(int) Member Function

Writes an **int** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(int i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An **int** to write. |

### WriteLine(long) Member Function

Writes a **long** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A **long** to write. |

### WriteLine(sbyte) Member Function

Writes an **sbyte** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(sbyte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | sbyte | An **sbyte** to write. |

### WriteLine(short) Member Function

Writes a **short** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A **short** to write. |

### WriteLine(uchar) Member Function

Writes a **uchar** followed by a newline to the output stream.

#### Syntax

```
public abstract void WriteLine(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A **uchar** to write. |

### WriteLine(uint) Member Function

Writes a **uint** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(uint i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | uint | A **uint** to write. |

### WriteLine(ulong) Member Function

Writes a **ulong** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | A **ulong** to write. |

### WriteLine(ushort) Member Function

Writes a **ushort** followed by a new line to the output stream.

#### Syntax

```
public abstract void WriteLine(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | A **ushort** to write. |

### WriteLine(wchar) Member Function

Writes a **wchar** followed by a newline to the output stream.

#### Syntax

```
public abstract void WriteLine(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A **wchar** to write. |

**~OutputStream() Member Function**

Destructor.

**Syntax**

```
public ~OutputStream();
```

### 5.10.16.2 Nonmember Functions

| Function | Description |
|---|---|
| operator<<(System.IO.OutputStream&, System.Date) | Writes the given date to the given output stream. |
| operator<<(System.IO.OutputStream&, System.EndLine) | Writes end line character to the given output stream. |
| operator<<(System.IO.OutputStream&, bool) | Writes the given Boolean value to the given output stream. |
| operator<<(System.IO.OutputStream&, byte) | Writes the given byte to the given output stream. |
| operator<<(System.IO.OutputStream&, char) | Writes the given character to the given output stream. |
| operator<<(System.IO.OutputStream&, const C&) | Writes the given forward container containing integers to the given output stream. |
| operator<<(System.IO.OutputStream&, const System.String<char>&) | Writes the given string to the given output stream. |
| operator<<(System.IO.OutputStream&, const System.String<uchar>&) | Writes the given UTF-32 string to the given output stream. |
| operator<<(System.IO.OutputStream&, const System.String<wchar>&) | Writes the given UTF-16 string to the given output stream. |
| operator<<(System.IO.OutputStream&, const char*) | Writes the given C-style string to the given output stream. |
| operator<<(System.IO.OutputStream&, double) | Writes the given double to the given output stream. |
| operator<<(System.IO.OutputStream&, float) | Writes the given float to the given output stream. |
| operator<<(System.IO.OutputStream&, int) | Writes the given int to the given output stream. |
| operator<<(System.IO.OutputStream&, long) | Writes the given long to the given output stream. |
| operator<<(System.IO.OutputStream&, sbyte) | Writes the given sbyte to the given output stream. |
| operator<<(System.IO.OutputStream&, short) | Writes the given short to the given output stream. |

| | |
|---|---|
| operator<<(System.IO.OutputStream&, uchar) | Writes the given character to the given output stream. |
| operator<<(System.IO.OutputStream&, uint) | Writes the given uint to the given output stream. |
| operator<<(System.IO.OutputStream&, ulong) | Writes the given ulong to given output stream. |
| operator<<(System.IO.OutputStream&, ushort) | Writes the given ushort to the given output stream. |
| operator<<(System.IO.OutputStream&, wchar) | Writes the given character to the given output stream. |

### operator<<(System.IO.OutputStream&, System.Date) Function

Writes the given date to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, System.Date
date);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| date | System.Date | A date to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, System.EndLine) Function

Writes end line character to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, System.EndLine
__parameter1);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| __parameter1 | System.EndLine | |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, bool) Function

Writes the given Boolean value to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| b | bool | A Boolean to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, byte) Function

Writes the given byte to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| b | byte | A byte to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, char) Function

Writes the given character to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| c | char | A character to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, const C&) Function

Writes the given forward container containing integers to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, const C& c);
```

## Constraint

int and C.ValueType is C.ValueType

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| c | const C& | A forward container. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, const System.String<char>&) Function

Writes the given string to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, const System.String<char>&
str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| str | const System.String<char>& | A string to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, const System.String<uchar>&) Function

Writes the given UTF-32 string to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, const System.String<uchar>
str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| str | const System.String<uchar>& | A string to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, const System.String<wchar>&) Function

Writes the given UTF-16 string to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, const System.String<wchar>
str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| str | const System.String<wchar>& | A string to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, const char*) Function

Writes the given C-style string to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, const char*
str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| str | const char* | A C-style string. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, double) Function

Writes the given double to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| d | double | A double to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, float) Function

Writes the given float to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| f | float | A float to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, int) Function

Writes the given int to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, int i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| i | int | An int to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, long) Function

Writes the given long to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | A output stream. |
| l | long | A long to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, sbyte) Function

Writes the given sbyte to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, sbyte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| b | sbyte | An sbyte to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, short) Function

Writes the given short to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, short x);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| x | short | A short to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, uchar) Function

Writes the given character to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| c | uchar | A character to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, uint) Function

Writes the given uint to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, uint u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| u | uint | A uint to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, ulong) Function

Writes the given ulong to given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| u | ulong | A ulong to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, ushort) Function

Writes the given ushort to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| u | ushort | A ushort to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

### operator<<(System.IO.OutputStream&, wchar) Function

Writes the given character to the given output stream.

#### Syntax

```
public System.IO.OutputStream& operator<<(System.IO.OutputStream& s, wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | System.IO.OutputStream& | An output stream. |
| c | wchar | A character to write. |

#### Returns

System.IO.OutputStream&

Returns a reference to the output stream.

## 5.10.17 OutputStringStream Class

A class for writing to a string.

### Syntax

```
public class OutputStringStream;
```

### Base Class

System.IO.OutputStream

#### 5.10.17.1 Member Functions

| Member Function | Description |
|---|---|
| OutputStringStream() | Default constructor. Write to an empty string. |
| OutputStringStream(System.IO.-OutputStringStream&&) | Move constructor. |
| operator=(System.IO.OutputStringStream&&) | Move assignment. |
| GetStr() const | Returns the contained string. |
| OutputStringStream(const System.String<-char>&) | Constructor. Write to the end of the given string. |
| SetStr(const System.String<char>&) | Sets the contained string. |
| Write(bool) | Writes a Boolean value to the end of the contained string. |
| Write(byte) | Writes a **byte** to the end of the contained string. |
| Write(char) | Writes a character to the endf of the contained string. |
| Write(const System.String<char>&) | Writes a string to the end of the contained string. |
| Write(const System.String<uchar>&) | Writes a UTF-32 string to the end of the contained string. |
| Write(const System.String<wchar>&) | Writes a UTF-16 string to the end of the contained string. |
| Write(const char*) | Writes a C-style string to the end of the contained string. |
| Write(double) | Writes a **double** to the end of the contained string. |

| | |
|---|---|
| Write(float) | Writes a **float** to the end of the contained string. |
| Write(int) | Writes an **int** to the end of the contained string. |
| Write(long) | Writes a **long** to the end of the contained string. |
| Write(sbyte) | Writes an **sbyte** to the end of the contained string. |
| Write(short) | Writes a **short** to the end of the contained string. |
| Write(uchar) | Writes a **uchar** to the end of the contained string. |
| Write(uint) | Writes a **uint** to the end of the contained string. |
| Write(ulong) | Writes a **ulong** to the end of the contained string. |
| Write(ushort) | Writes a **ushort** to the end of the contained string. |
| Write(wchar) | Writes a **wchar** to the end of the contained string. |
| WriteLine() | Writes a new line to the end of the contained string. |
| WriteLine(bool) | Writes a Boolean followed by a new line to the end of the contained string. |
| WriteLine(byte) | Writes a **byte** followed by a new line to the end of the contained string. |
| WriteLine(char) | Writes a character followed by a new line to the end of the contained string. |
| WriteLine(const System.String<char>&) | Writes a string followed by a new line to the end of the contained string. |
| WriteLine(const System.String<uchar>&) | Writes a UTF-32 string followed by a new line to the end of the contained string. |
| WriteLine(const System.String<wchar>&) | Writes a UTF-16 string followed by a new line to the end of the contained string. |
| WriteLine(const char*) | Writes a C-style string followed by a new line to the end of the contained string. |
| WriteLine(double) | Writes a **double** followed by a new line to the end of the contained string. |

| | |
|---|---|
| WriteLine(float) | Writes a **float** followed by a new line to the end of the contained string. |
| WriteLine(int) | Writes an **int** followed by a new line to the end of the contained string. |
| WriteLine(long) | Writes a **long** followed by a new line to the end of the contained string. |
| WriteLine(sbyte) | Writes an **sbyte** followed by a new line to the end of the contained string. |
| WriteLine(short) | Writes a **short** followed by a new line to the end of the contained string. |
| WriteLine(uchar) | Writes a **uchar** followed by a new line to the end of the contained string. |
| WriteLine(uint) | Writes a **uint** followed by a new line to the end of the contained string. |
| WriteLine(ulong) | Writes a **ulong** followed by a new line to the end of the contained string. |
| WriteLine(ushort) | Writes a **ushort** followed by a new line to the end of the contained string. |
| WriteLine(wchar) | Writes a **wchar** followed by a new line to the end of the contained string. |
| ~OutputStringStream() | Destructor. |

### OutputStringStream() Member Function

Default constructor. Write to an empty string.

#### Syntax

```
public OutputStringStream();
```

### OutputStringStream(System.IO.OutputStringStream&&) Member Function

Move constructor.

#### Syntax

```
public OutputStringStream(System.IO.OutputStringStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.OutputStringStream&& | |

### operator=(System.IO.OutputStringStream&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.IO.OutputStringStream&& __parameter0);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| __parameter0 | System.IO.OutputStringStream&& | |

### GetStr() const Member Function

Returns the contained string.

#### Syntax

```
public const System.String<char>& GetStr() const;
```

#### Returns

const System.String<char>&

Returns the contained string.

### OutputStringStream(const System.String<char>&) Member Function

Constructor. Write to the end of the given string.

#### Syntax

```
public OutputStringStream(const System.String<char>& str_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| str_ | const System.String<char>& | A string to write to. |

### SetStr(const System.String<char>&) Member Function

Sets the contained string.

#### Syntax

```
public void SetStr(const System.String<char>& str_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| str_ | const System.String<char>& | A string to write to. |

### Write(bool) Member Function

Writes a Boolean value to the end of the contained string.

#### Syntax

```
public void Write(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean value to write. |

### Write(byte) Member Function

Writes a **byte** to the end of the contained string.

### Syntax

```
public void Write(byte b);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A **byte** to write. |

### Write(char) Member Function

Writes a character to the endf of the contained string.

### Syntax

```
public void Write(char c);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### Write(const System.String<char>&) Member Function

Writes a string to the end of the contained string.

#### Syntax

```
public void Write(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to write. |

### Write(const System.String<uchar>&) Member Function

Writes a UTF-32 string to the end of the contained string.

#### Syntax

```
public void Write(const System.String<uchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to write. |

### Write(const System.String<wchar>&) Member Function

Writes a UTF-16 string to the end of the contained string.

#### Syntax

```
public void Write(const System.String<wchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to write. |

### Write(const char*) Member Function

Writes a C-style string to the end of the contained string.

#### Syntax

```
public void Write(const char* s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### Write(double) Member Function

Writes a **double** to the end of the contained string.

#### Syntax

```
public void Write(double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A **double** to write. |

### Write(float) Member Function

Writes a **float** to the end of the contained string.

#### Syntax

```
public void Write(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A **float** to write. |

### Write(int) Member Function

Writes an **int** to the end of the contained string.

### Syntax

```
public void Write(int i);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An **int** to write. |

### Write(long) Member Function

Writes a **long** to the end of the contained string.

#### Syntax

```
public void Write(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A **long** to write. |

### Write(sbyte) Member Function

Writes an **sbyte** to the end of the contained string.

### Syntax

```
public void Write(sbyte b);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| b | sbyte | An **sbyte** to write. |

### Write(short) Member Function

Writes a **short** to the end of the contained string.

#### Syntax

```
public void Write(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A **short** to write. |

### Write(uchar) Member Function

Writes a **uchar** to the end of the contained string.

#### Syntax

```
public void Write(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A **uchar** to write. |

### Write(uint) Member Function

Writes a **uint** to the end of the contained string.

#### Syntax

```
public void Write(uint i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | uint | A **uint** to write. |

### Write(ulong) Member Function

Writes a **ulong** to the end of the contained string.

#### Syntax

```
public void Write(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | A **ulong** to write. |

### Write(ushort) Member Function

Writes a **ushort** to the end of the contained string.

#### Syntax

```
public void Write(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | A **ushort** to write. |

### Write(wchar) Member Function

Writes a **wchar** to the end of the contained string.

#### Syntax

```
public void Write(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A **wchar** to write. |

### WriteLine() Member Function

Writes a new line to the end of the contained string.

### Syntax

```
public void WriteLine();
```

### WriteLine(bool) Member Function

Writes a Boolean followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(bool b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | bool | A Boolean to write. |

### WriteLine(byte) Member Function

Writes a **byte** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(byte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | byte | A **byte** to write. |

### WriteLine(char) Member Function

Writes a character followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to write. |

### WriteLine(const System.String<char>&) Member Function

Writes a string followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to write. |

### WriteLine(const System.String<uchar>&) Member Function

Writes a UTF-32 string followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(const System.String<uchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<uchar>& | A string to write. |

### WriteLine(const System.String<wchar>&) Member Function

Writes a UTF-16 string followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(const System.String<wchar>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<wchar>& | A string to write. |

### WriteLine(const char*) Member Function

Writes a C-style string followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(const char* s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const char* | A C-style string to write. |

### WriteLine(double) Member Function

Writes a **double** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(double d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | double | A **double** to write. |

### WriteLine(float) Member Function

Writes a **float** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(float f);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| f | float | A **float** to write. |

### WriteLine(int) Member Function

Writes an **int** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(int i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | An **int** to write. |

### WriteLine(long) Member Function

Writes a **long** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(long l);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| l | long | A **long** to write. |

### WriteLine(sbyte) Member Function

Writes an **sbyte** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(sbyte b);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| b | sbyte | An **sbyte** to write. |

### WriteLine(short) Member Function

Writes a **short** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(short s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | short | A **short** to write. |

### WriteLine(uchar) Member Function

Writes a **uchar** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A **uchar** to write. |

### WriteLine(uint) Member Function

Writes a **uint** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(uint i);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| i | uint | A **uint** to write. |

### WriteLine(ulong) Member Function

Writes a **ulong** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(ulong u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ulong | A **ulong** to write. |

### WriteLine(ushort) Member Function

Writes a **ushort** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(ushort u);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| u | ushort | A **ushort** to write. |

### WriteLine(wchar) Member Function

Writes a **wchar** followed by a new line to the end of the contained string.

#### Syntax

```
public void WriteLine(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A **wchar** to write. |

### ~OutputStringStream() Member Function

Destructor.

#### Syntax

```
public ~OutputStringStream();
```

## 5.10.18 Path Class

A static class for manipulating paths.

### Syntax

```
public static class Path;
```

### 5.10.18.1 Member Functions

| Member Function | Description |
|---|---|
| ChangeExtension(const System.String<char>&-, const System.String<char>&) const | Changes an extension of a path. |
| Combine(const System.String<char>&, const System.String<char>&) const | Combines to paths. |
| GetDirectoryName(const System.String<char>-&) const | Returns the directory name part of a path. |
| GetExtension(const System.String<char>&) const | Returns the extension of a path. |
| GetFileName(const System.String<char>&) const | Returns the file name part of a path. |
| GetFileNameWithoutExtension(const System.-String<char>&) const | Returns the file name without an extension part of a path. |
| GetParent(const System.String<char>&) const | Returns a given path with last component removed. |
| HasExtension(const System.String<char>&) const | Returns true, if the given path has an extension, false otherwise. |
| IsAbsolute(const System.String<char>&) const | Returns true if the given path is absolute, false otherwise. |
| IsRelative(const System.String<char>&) const | Returns true if the given path is relative, false otherwise. |
| MakeCanonical(const System.String<char>&) const | Returns a canonical representation of a path. |

### ChangeExtension(const System.String<char>&, const System.String<char>&) const Member Function

Changes an extension of a path.

#### Syntax

```
public static System.String<char> ChangeExtension(const System.String<char>& path,
const System.String<char>& extension) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |
| extension | const System.String<char>& | A new extension. |

#### Returns

System.String<char>

A path with a new extension.

#### Remarks

If new extension is empty, returns a path with extension removed. Otherwise, if path has no extension, returns a path with the new extension appended. Otherwise, returns a path with the new extension replaced. New extension can have '.' in it or not.

### Combine(const System.String<char>&, const System.String<char>&) const Member Function

Combines to paths.

#### Syntax

```
public static System.String<char> Combine(const System.String<char>& path1, const
System.String<char>& path2) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path1 | const System.String<char>& | The first path. |
| path2 | const System.String<char>& | The second path. |

#### Returns

System.String<char>

Returns the first path combined with the second path.

#### Remarks

If the first path is empty, returns the second path. Otherwise, if the second path is empty, returns the first path. Otherwise, if the second path is absolute, returns the second path. Otherwise returns the first path and the second path separated by the '/' character.

### GetDirectoryName(const System.String<char>&) const Member Function

Returns the directory name part of a path.

#### Syntax

```
public static System.String<char> GetDirectoryName(const System.String<char>& path)
const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

System.String<char>

A path with last component removed.

#### Remarks

If the path parameter is empty, returns empty string. Otherwise, if the path consists of alphabetical letter, a colon and a slash, returns empty string. Otherwise, if the path has a '/' character, returns a path with the last '/' character and characters following it removed. Otherwise returns an empty string.

### GetExtension(const System.String<char>&) const Member Function

Returns the extension of a path.

#### Syntax

```
public static System.String<char> GetExtension(const System.String<char>& path)
const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

System.String<char>

Returns the extension of the path.

#### Remarks

If the path contains a '.' character but it also contains '/' character after the last '.' character, an empty string is returned. Otherwise, if the path contains a '.' character, returns a substring containing the last '.' character and characters following it. Otherwise returns an empty string.

### GetFileName(const System.String<char>&) const Member Function

Returns the file name part of a path.

#### Syntax

```
public static System.String<char> GetFileName(const System.String<char>& path) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

System.String<char>

Returns a path with extension removed.

#### Remarks

If the given path is empty, or the last character of it is '/' or ':', returns empty string. Otherwise, if the path contains a '/' character, returns the characters following the last '/' character. Otherwise returns the path.

### GetFileNameWithoutExtension(const System.String<char>&) const Member Function

Returns the file name without an extension part of a path.

#### Syntax

```
public static System.String<char> GetFileNameWithoutExtension(const System.String<char>&
path) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

System.String<char>

Returns a file name without an extension.

#### Remarks

First gets the file name part of the path. If the file name has an extension, returns the file name with extension removed. Otherwise returns the file name.

### GetParent(const System.String<char>&) const Member Function

Returns a given path with last component removed.

#### Syntax

```
public static System.String<char> GetParent(const System.String<char>& path) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

System.String<char>

Returns a given path with last component removed.

### HasExtension(const System.String<char>&) const Member Function

Returns true, if the given path has an extension, false otherwise.

#### Syntax

```
public static bool HasExtension(const System.String<char>& path) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

bool

Returns true, if the given path has an extension, false otherwise.

#### Remarks

If the path has no '.' character or the path ends with a '.' character, returns false. Otherwise, if the path has a '/' or ':' character after the last '.' character, returns false. Otherwise returns true.

### IsAbsolute(const System.String<char>&) const Member Function

Returns true if the given path is absolute, false otherwise.

#### Syntax

```
public static bool IsAbsolute(const System.String<char>& path) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

bool

Returns true if the given path is absolute, false otherwise.

#### Remarks

If the given path is empty, returns false. Otherwise, if the given path begins with the '/' character, returns true. Otherwise, if the given path begins with an alphabetical letter followed by a ':' character followed by a '/' character, returns true. Otherwise returns false.

### IsRelative(const System.String<char>&) const Member Function

Returns true if the given path is relative, false otherwise.

#### Syntax

```
public static bool IsRelative(const System.String<char>& path) const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

bool

Returns true if the given path is relative, false otherwise.

#### Remarks

Returns the complement of the value returned by the System.IO.Path.IsAbsolute.System.String.const.ref function for the given path.

### MakeCanonical(const System.String<char>&) const Member Function

Returns a canonical representation of a path.

#### Syntax

```
public static System.String<char> MakeCanonical(const System.String<char>& path)
const;
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

System.String<char>

Returns a canonical representation of the given path.

#### Remarks

First replaces each \ character of the path with the '/' character. Then, if the path consists of an alphabetical letter followed by the ':' character followed by the '/' character returns the path. Otherwise, if the path consists of a '/' character, returns the path. Otherwise, if the path ends with a '/' character, returns the path with last '/' character removed. Otherwise returns the path.

## 5.11 Functions

| Function | Description |
|---|---|
| CreateDirectories(const System.String<char>&) | Creates all directories along the given directory path. |
| DirectoryExists(const System.String<char>&) | Returns true if a directory with a given path name exists, false otherwise. |
| FileContentsEqual(const System.String<char>-&, const System.String<char>&) | Returns true if the given files have equal content, false otherwise. |
| FileExists(const System.String<char>&) | Returns true if a file with a given path name exists, false otherwise. |
| GetCurrentWorkingDirectory() | Returns a path to the current working directory. |
| GetFullPath(const System.String<char>&) | Returns an absolute path corresponding to the given absolute or relative path. |
| GetPathToExecutable() | Returns a path to the currently executing executable. |
| LastWriteTimeLess(const System.String<char>-&, const System.String<char>&) | Returns true if the modification timestamp of filePath1 is less than the modification timestamp of filePath2, false otherwise. |
| PathExists(const System.String<char>&) | Returns true if the given path exists, false otherwise. |
| ReadFile(const System.String<char>&) | Reads a file with the given name into a string and returns it. |
| RemoveFile(const System.String<char>&) | Removes the given file. |

### 5.11.19 CreateDirectories(const System.String<char>&) Function

Creates all directories along the given directory path.

#### Syntax

```
public void CreateDirectories(const System.String<char>& directoryPath);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| directoryPath | const System.String<char>& | A directory path. |

### 5.11.20 DirectoryExists(const System.String<char>&) Function

Returns true if a directory with a given path name exists, false otherwise.

#### Syntax

```
public bool DirectoryExists(const System.String<char>& directoryPath);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| directoryPath | const System.String<char>& | A path to test. |

#### Returns

bool

Returns true if a directory with a given path name exists, false otherwise.

### 5.11.21 FileContentsEqual(const System.String<char>&, const System.String<char>&) Function

Returns true if the given files have equal content, false otherwise.

#### Syntax

```
public bool FileContentsEqual(const System.String<char>& fileName1, const System.String<char>&
fileName2);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| fileName1 | const System.String<char>& | The first file. |
| fileName2 | const System.String<char>& | The second file. |

## Returns

bool

Returns true if the given files have equal content, false otherwise.

### 5.11.22 FileExists(const System.String<char>&) Function

Returns true if a file with a given path name exists, false otherwise.

#### Syntax

```
public bool FileExists(const System.String<char>& filePath);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| filePath | const System.String<char>& | A path to test. |

#### Returns

bool

Returns true if a file with a given path name exists, false otherwise.

### 5.11.23 GetCurrentWorkingDirectory() Function

Returns a path to the current working directory.

#### Syntax

```
public System.String<char> GetCurrentWorkingDirectory();
```

#### Returns

System.String<char>

Returns a path to the current working directory.

### 5.11.24 GetFullPath(const System.String<char>&) Function

Returns an absolute path corresponding to the given absolute or relative path.

#### Syntax

```
public System.String<char> GetFullPath(const System.String<char>& path);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

System.String<char>

Returns an absolute path corresponding to the given absolute or relative path.

#### Remarks

If the given path is relative, prefixes it with the path to current working directory.

### 5.11.25 GetPathToExecutable() Function

Returns a path to the currently executing executable.

#### Syntax

```
public System.String<char> GetPathToExecutable();
```

#### Returns

System.String<char>

Returns a path to the currently executing executable.

### 5.11.26 LastWriteTimeLess(const System.String<char>&, const System.String<char>&) Function

Returns true if the modification timestamp of filePath1 is less than the modification timestamp of filePath2, false otherwise.

#### Syntax

```
public bool LastWriteTimeLess(const System.String<char>& filePath1, const System.String<char>&
filePath2);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| filePath1 | const System.String<char>& | The first path. |
| filePath2 | const System.String<char>& | The second path. |

#### Returns

bool

Returns true if the modification timestamp of filePath1 is less than the modification timestamp of filePath2, false otherwise.

### 5.11.27 PathExists(const System.String<char>&) Function

Returns true if the given path exists, false otherwise.

#### Syntax

```
public bool PathExists(const System.String<char>& path);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| path | const System.String<char>& | A path. |

#### Returns

bool

Returns true if the given path exists, false otherwise.

### 5.11.28 ReadFile(const System.String<char>&) Function

Reads a file with the given name into a string and returns it.

#### Syntax

```
public System.String<char> ReadFile(const System.String<char>& fileName);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| fileName | const System.String<char>& | The name of the file to read. |

#### Returns

System.String<char>

Returns the contents of the given file.

### 5.11.29 RemoveFile(const System.String<char>&) Function

Removes the given file.

#### Syntax

```
public void RemoveFile(const System.String<char>& filePath);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| filePath | const System.String<char>& | A path. |

## 5.12 Enumerations

| Enumeration | Description |
|---|---|
| FileMode | Open mode for FileByteStream. |
| OpenMode | An open mode for a binary file stream. |

#### 5.12.29.1 FileMode Enumeration

Open mode for FileByteStream.

**Enumeration Constants**

| Constant | Value | Description |
|---|---|---|
| append | 0 | Opens the file for appending to the end of it. |
| create | 1 | Creates a file if it does not exists, or truncates it to zero bytes if exists. The file is opened for writing. |
| open | 2 | Opens the file for reading. |

#### 5.12.29.2 OpenMode Enumeration

An open mode for a binary file stream.

**Enumeration Constants**

| Constant | Value | Description |
|---|---|---|
| readOnly | 0 | Open the file with read only access. |
| readWrite | 2 | Open the file with read and write access. |
| writeOnly | 1 | Open the file with write only access. Truncates the file if it exists. |

# 6 System.Meta Namespace

Contains instrinsic type predicate functions.

## 6.13 Functions

| Function | Description |
|---|---|
| IsBasicType<T0>() | Returns true if T0 is a basic type, false otherwise. |
| IsBoolType<T0>() | Returns true if T0 is bool, false otherwise. |
| IsByteType<T0>() | Returns true if T0 is byte, false otherwise. |
| IsCharType<T0>() | Returns true if T0 is char, false otherwise. |
| IsCharacterType<T0>() | Returns true if T0 is char, wchar or uchar, false otherwise. |
| IsClassDelegateType<T0>() | Returns true if T0 is a class delegate type, false otherwise. |
| IsClassType<T0>() | Returns true if T0 is a class type, false otherwise. |
| IsConstType<T0>() | Returns true if T0 is a const type, false otherwise. |
| IsDelegateType<T0>() | Returns true if T0 is a delegate type, false otherwise. |
| IsDoubleType<T0>() | Returns true if T0 is double, false otherwise. |
| IsEnumType<T0>() | Returns true if T0 is enumerated type, false otherwise. |
| IsFloatType<T0>() | Returns true if T0 is float, false otherwise. |
| IsFloatingPointType<T0>() | Returns true if T0 is float or double, false otherwise. |
| IsIntType<T0>() | Returns true if T0 is int, false otherwise. |
| IsIntegerType<T0>() | Returns true if T0 is sbyte, byte, short ushort, int, uint, long, or ulong, false otherwise. |
| IsLongType<T0>() | Returns true if T0 is long, false otherwise. |
| IsLvalueReferenceType<T0>() | Returns true if T0 is an lvalue reference type, false otherwise. |
| IsNonBasicType<T0>() | Returns true if T0 is not a basic type, false otherwise. |

| | |
|---|---|
| IsNonBoolType<T0>() | Returns true if T0 is not bool, false otherwise. |
| IsNonByteType<T0>() | Returns true if T0 is not byte, false otherwise. |
| IsNonCharType<T0>() | Returns true if T0 is not char, false otherwise. |
| IsNonCharacterType<T0>() | Returns true if T0 is not char, wchar or uchar, false otherwise. |
| IsNonClassDelegateType<T0>() | Returns true if T0 is not a class delegate type, false otherwise. |
| IsNonClassType<T0>() | Returns true if T0 is not a class type, false otherwise. |
| IsNonConstType<T0>() | Returns true if T0 is not a const type, false otherwise. |
| IsNonDelegateType<T0>() | Returns true if T0 is not a delegate type, false otherwise. |
| IsNonDoubleType<T0>() | Returns true if T0 is not double, false otherwise. |
| IsNonEnumType<T0>() | Returns true if T0 is not an enumerated type, false otherwise. |
| IsNonFloatType<T0>() | Returns true if T0 is not float, false otherwise. |
| IsNonFloatingPointType<T0>() | Returns true if T0 is not float or double, false otherwise. |
| IsNonIntType<T0>() | Returns true if T0 is not int, false otherwise. |
| IsNonIntegerType<T0>() | Returns true if T0 is not sbyte, byte, short, ushort, int, uint, long, ulong, false otherwise. |
| IsNonLongType<T0>() | Returns true if T0 is not long, false otherwise. |
| IsNonLvalueReferenceType<T0>() | Returns true if T0 is not an lvalue reference type, false otherwise. |
| IsNonPointerType<T0>() | Returns true if T0 is not a pointer type, false otherwise. |
| IsNonPolymorphic<T0>() | Returns true if T0 is not polymorphic type, false otherwise. |

| | |
|---|---|
| IsNonReferenceType<T0>() | Returns true if T0 is not an lvalue or rvalue reference type, false otherwise. |
| IsNonRvalueReferenceType<T0>() | Returns true if T0 is not an rvalue reference type, false otherwise. |
| IsNonSByteType<T0>() | Returns true if T0 is not sbyte, false otherwise. |
| IsNonShortType<T0>() | Returns true if T0 is not short, false otherwise. |
| IsNonSignedType<T0>() | Returns true if T0 is not sbyte, short, int, long, false otherwise. |
| IsNonUCharType<T0>() | Returns true if T0 is not uchar, false otherwise. |
| IsNonUIntType<T0>() | Returns true if T0 is not uint, false otherwise. |
| IsNonULongType<T0>() | Returns true if T0 is not ulong, false otherwise. |
| IsNonUShortType<T0>() | Returns true if T0 is not ushort, false otherwise. |
| IsNonUnsignedType<T0>() | Returns true if T0 is not byte, ushort, uint, ulong, false otherwise. |
| IsNonVoidType<T0>() | Returns true if T0 is not void, false otherwise. |
| IsNonWCharType<T0>() | Returns true if T0 is not wchar, false otherwise. |
| IsPointerType<T0>() | Returns true if T0 is a pointer type, false otherwise. |
| IsPolymorphic<T0>() | Returns true if T0 is a polymorphic type, false otherwise. |
| IsReferenceType<T0>() | Returns true if T0 is an lvalue or rvalue reference type, false otherwise. |
| IsRvalueReferenceType<T0>() | Returns true if T0 is an rvalue reference type, false otherwise. |
| IsSByteType<T0>() | Returns true if T0 is sbyte, false otherwise. |
| IsShortType<T0>() | Returns true if T0 is short, false otherwise. |
| IsSignedType<T0>() | Returns true if T0 is sbyte, short, int, long, false otherwise. |

| | |
|---|---|
| IsUCharType<T0>() | Returns true if T0 is uchar, false otherwise. |
| IsUIntType<T0>() | Returns true if T0 is uint, false otherwise. |
| IsULongType<T0>() | Returns true if T0 is ulong, false otherwise. |
| IsUShortType<T0>() | Returns true if T0 is ushort, false otherwise. |
| IsUnsignedType<T0>() | Returns true if T0 is byte, ushort, uint, ulong, false otherwise. |
| IsVoidType<T0>() | Returns true if T0 is void, false otherwise. |
| IsWCharType<T0>() | Returns true if T0 is wchar, false otherwise. |

### 6.13.1 IsBasicType<T0>() Function

Returns true if T0 is a basic type, false otherwise.

#### Syntax

```
public bool IsBasicType<T0>();
```

#### Returns

bool

Returns true if T0 is a basic type, false otherwise.

#### Remarks

Basic types are: bool, char, wchar, uchar, void, sbyte, byte, short, ushort, int, uint, long, ulong, float and double.

### 6.13.2 IsBoolType<T0>() Function

Returns true if T0 is bool, false otherwise.

#### Syntax

```
public bool IsBoolType<T0>();
```

#### Returns

bool

Returns true if T0 is bool, false otherwise.

### 6.13.3 IsByteType<T0>() Function

Returns true if T0 is byte, false otherwise.

#### Syntax

```
public bool IsByteType<T0>();
```

#### Returns

bool

Returns true if T0 is byte, false otherwise.

### 6.13.4 IsCharType<T0>() Function

Returns true if T0 is char, false otherwise.

#### Syntax

```
public bool IsCharType<T0>();
```

#### Returns

bool

Returns true if T0 is char, false otherwise.

### 6.13.5 IsCharacterType<T0>() Function

Returns true if T0 is char, wchar or uchar, false otherwise.

**Syntax**

```
public bool IsCharacterType<T0>();
```

#### Returns

bool

Returns true if T0 is char, wchar or uchar, false otherwise.

### 6.13.6 IsClassDelegateType<T0>() Function

Returns true if T0 is a class delegate type, false otherwise.

**Syntax**

```
public bool IsClassDelegateType<T0>();
```

## Returns

bool

Returns true if T0 is a class delegate type, false otherwise.

### 6.13.7 IsClassType<T0>() Function

Returns true if T0 is a class type, false otherwise.

#### Syntax

```
public bool IsClassType<T0>();
```

#### Returns

bool

Returns true if T0 is a class type, false otherwise.

### 6.13.8 IsConstType<T0>() Function

Returns true if T0 is a const type, false otherwise.

#### Syntax

```
public bool IsConstType<T0>();
```

#### Returns

bool

Returns true if T0 is a const type, false otherwise.

### 6.13.9 IsDelegateType<T0>() Function

Returns true if T0 is a delegate type, false otherwise.

**Syntax**

```
public bool IsDelegateType<T0>();
```

## Returns

bool

Returns true if T0 is a delegate type, false otherwise.

### 6.13.10 IsDoubleType<T0>() Function

Returns true if T0 is double, false otherwise.

#### Syntax

```
public bool IsDoubleType<T0>();
```

#### Returns

bool

Returns true if T0 is double, false otherwise.

### 6.13.11 IsEnumType<T0>() Function

Returns true if T0 is enumerated type, false otherwise.

#### Syntax

```
public bool IsEnumType<T0>();
```

#### Returns

bool

Returns true if T0 is enumerated type, false otherwise.

### 6.13.12 IsFloatType<T0>() Function

Returns true if T0 is float, false otherwise.

#### Syntax

```
public bool IsFloatType<T0>();
```

#### Returns

bool

Returns true if T0 is float, false otherwise.

### 6.13.13 IsFloatingPointType<T0>() Function

Returns true if T0 is float or double, false otherwise.

#### Syntax

```
public bool IsFloatingPointType<T0>();
```

#### Returns

bool

Returns true if T0 is float or double, false otherwise.

### 6.13.14 IsIntType<T0>() Function

Returns true if T0 is int, false otherwise.

#### Syntax

```
public bool IsIntType<T0>();
```

#### Returns

bool

Returns true if T0 is int, false otherwise.

### 6.13.15 IsIntegerType<T0>() Function

Returns true if T0 is sbyte, byte, short ushort, int, uint, long, or ulong, false otherwise.

#### Syntax

```
public bool IsIntegerType<T0>();
```

#### Returns

bool

Returns true if T0 is sbyte, byte, short ushort, int, uint, long, or ulong, false otherwise.

### 6.13.16 IsLongType<T0>() Function

Returns true if T0 is long, false otherwise.

#### Syntax

```
public bool IsLongType<T0>();
```

#### Returns

bool

Returns true if T0 is long, false otherwise.

### 6.13.17 IsLvalueReferenceType<T0>() Function

Returns true if T0 is an lvalue reference type, false otherwise.

#### Syntax

```
public bool IsLvalueReferenceType<T0>();
```

#### Returns

bool

Returns true if T0 is an lvalue reference type, false otherwise.

### 6.13.18 IsNonBasicType<T0>() Function

Returns true if T0 is not a basic type, false otherwise.

#### Syntax

```
public bool IsNonBasicType<T0>();
```

#### Returns

bool

Returns true if T0 is not a basic type, false otherwise.

### 6.13.19 IsNonBoolType<T0>() Function

Returns true if T0 is not bool, false otherwise.

#### Syntax

```
public bool IsNonBoolType<T0>();
```

#### Returns

bool

Returns true if T0 is not bool, false otherwise.

### 6.13.20 IsNonByteType<T0>() Function

Returns true if T0 is not byte, false otherwise.

#### Syntax

```
public bool IsNonByteType<T0>();
```

#### Returns

bool

Returns true if T0 is not byte, false otherwise.

### 6.13.21 IsNonCharType<T0>() Function

Returns true if T0 is not char, false otherwise.

#### Syntax

```
public bool IsNonCharType<T0>();
```

#### Returns

bool

Returns true if T0 is not char, false otherwise.

### 6.13.22 IsNonCharacterType<T0>() Function

Returns true if T0 is not char, wchar or uchar, false otherwise.

#### Syntax

```
public bool IsNonCharacterType<T0>();
```

#### Returns

bool

Returns true if T0 is not char, wchar or uchar, false otherwise.

### 6.13.23 IsNonClassDelegateType<T0>() Function

Returns true if T0 is not a class delegate type, false otherwise.

**Syntax**

```
public bool IsNonClassDelegateType<T0>();
```

## Returns

bool

Returns true if T0 is not a class delegate type, false otherwise.

### 6.13.24 IsNonClassType<T0>() Function

Returns true if T0 is not a class type, false otherwise.

#### Syntax

```
public bool IsNonClassType<T0>();
```

#### Returns

bool

Returns true if T0 is not a class type, false otherwise.

### 6.13.25 IsNonConstType<T0>() Function

Returns true if T0 is not a const type, false otherwise.

#### Syntax

```
public bool IsNonConstType<T0>();
```

#### Returns

bool

Returns true if T0 is not a const type, false otherwise.

### 6.13.26 IsNonDelegateType<T0>() Function

Returns true if T0 is not a delegate type, false otherwise.

#### Syntax

```
public bool IsNonDelegateType<T0>();
```

#### Returns

bool

Returns true if T0 is not a delegate type, false otherwise.

### 6.13.27 IsNonDoubleType<T0>() Function

Returns true if T0 is not double, false otherwise.

#### Syntax

```
public bool IsNonDoubleType<T0>();
```

#### Returns

bool

Returns true if T0 is not double, false otherwise.

### 6.13.28 IsNonEnumType<T0>() Function

Returns true if T0 is not an enumerated type, false otherwise.

**Syntax**

```
public bool IsNonEnumType<T0>();
```

## Returns

bool

Returns true if T0 is not an enumerated type, false otherwise.

### 6.13.29 IsNonFloatType<T0>() Function

Returns true if T0 is not float, false otherwise.

#### Syntax

```
public bool IsNonFloatType<T0>();
```

#### Returns

bool

Returns true if T0 is not float, false otherwise.

### 6.13.30 IsNonFloatingPointType<T0>() Function

Returns true if T0 is not float or double, false otherwise.

#### Syntax

```
public bool IsNonFloatingPointType<T0>();
```

#### Returns

bool

Returns true if T0 is not float or double, false otherwise.

### 6.13.31 IsNonIntType<T0>() Function

Returns true if T0 is not int, false otherwise.

#### Syntax

```
public bool IsNonIntType<T0>();
```

#### Returns

bool

Returns true if T0 is not int, false otherwise.

### 6.13.32 IsNonIntegerType<T0>() Function

Returns true if T0 is not sbyte, byte, short, ushort, int, uint, long, ulong, false otherwise.

#### Syntax

```
public bool IsNonIntegerType<T0>();
```

#### Returns

bool

Returns true if T0 is not sbyte, byte, short, ushort, int, uint, long, ulong, false otherwise.

### 6.13.33 IsNonLongType<T0>() Function

Returns true if T0 is not long, false otherwise.

#### Syntax

```
public bool IsNonLongType<T0>();
```

#### Returns

bool

Returns true if T0 is not long, false otherwise.

### 6.13.34 IsNonLvalueReferenceType<T0>() Function

Returns true if T0 is not an lvalue reference type, false otherwise.

**Syntax**

```
public bool IsNonLvalueReferenceType<T0>();
```

## Returns

bool

Returns true if T0 is not an lvalue reference type, false otherwise.

### 6.13.35 IsNonPointerType<T0>() Function

Returns true if T0 is not a pointer type, false otherwise.

#### Syntax

```
public bool IsNonPointerType<T0>();
```

#### Returns

bool

Returns true if T0 is not an lvalue reference type, false otherwise.

### 6.13.36 IsNonPolymorphic<T0>() Function

Returns true if T0 is not polymorphic type, false otherwise.

**Syntax**

```
public bool IsNonPolymorphic<T0>();
```

## Returns

bool

Returns true if T0 is not polymorphic type, false otherwise.

### 6.13.37 IsNonReferenceType<T0>() Function

Returns true if T0 is not an lvalue or rvalue reference type, false otherwise.

#### Syntax

```
public bool IsNonReferenceType<T0>();
```

#### Returns

bool

Returns true if T0 is not an lvalue or rvalue reference type, false otherwise.

### 6.13.38 IsNonRvalueReferenceType<T0>() Function

Returns true if T0 is not an rvalue reference type, false otherwise.

**Syntax**

```
public bool IsNonRvalueReferenceType<T0>();
```

## Returns

bool

Returns true if T0 is not an rvalue reference type, false otherwise.

### 6.13.39 IsNonSByteType<T0>() Function

Returns true if T0 is not sbyte, false otherwise.

#### Syntax

```
public bool IsNonSByteType<T0>();
```

#### Returns

bool

Returns true if T0 is not sbyte, false otherwise.

### 6.13.40 IsNonShortType<T0>() Function

Returns true if T0 is not short, false otherwise.

#### Syntax

```
public bool IsNonShortType<T0>();
```

#### Returns

bool

Returns true if T0 is not short, false otherwise.

### 6.13.41 IsNonSignedType<T0>() Function

Returns true if T0 is not sbyte, short, int, long, false otherwise.

**Syntax**

```
public bool IsNonSignedType<T0>();
```

## Returns

bool

Returns true if T0 is not sbyte, short, int, long, false otherwise.

### 6.13.42 IsNonUCharType<T0>() Function

Returns true if T0 is not uchar, false otherwise.

#### Syntax

```
public bool IsNonUCharType<T0>();
```

#### Returns

bool

Returns true if T0 is not uchar, false otherwise.

### 6.13.43 IsNonUIntType<T0>() Function

Returns true if T0 is not uint, false otherwise.

#### Syntax

```
public bool IsNonUIntType<T0>();
```

#### Returns

bool

Returns true if T0 is not uint, false otherwise.

### 6.13.44 IsNonULongType<T0>() Function

Returns true if T0 is not ulong, false otherwise.

#### Syntax

```
public bool IsNonULongType<T0>();
```

#### Returns

bool

Returns true if T0 is not ulong, false otherwise.

### 6.13.45 IsNonUShortType<T0>() Function

Returns true if T0 is not ushort, false otherwise.

#### Syntax

```
public bool IsNonUShortType<T0>();
```

#### Returns

bool

Returns true if T0 is not ushort, false otherwise.

### 6.13.46 IsNonUnsignedType<T0>() Function

Returns true if T0 is not byte, ushort, uint, ulong, false otherwise.

#### Syntax

```
public bool IsNonUnsignedType<T0>();
```

#### Returns

bool

Returns true if T0 is not byte, ushort, uint, ulong, false otherwise.

### 6.13.47 IsNonVoidType<T0>() Function

Returns true if T0 is not void, false otherwise.

#### Syntax

```
public bool IsNonVoidType<T0>();
```

#### Returns

bool

Returns true if T0 is not void, false otherwise.

### 6.13.48 IsNonWCharType<T0>() Function

Returns true if T0 is not wchar, false otherwise.

#### Syntax

```
public bool IsNonWCharType<T0>();
```

#### Returns

bool

Returns true if T0 is not wchar, false otherwise.

### 6.13.49 IsPointerType<T0>() Function

Returns true if T0 is a pointer type, false otherwise.

#### Syntax

```
public bool IsPointerType<T0>();
```

#### Returns

bool

Returns true if T0 is a pointer type, false otherwise.

### 6.13.50 IsPolymorphic<T0>() Function

Returns true if T0 is a polymorphic type, false otherwise.

#### Syntax

```
public bool IsPolymorphic<T0>();
```

#### Returns

bool

Returns true if T0 is a polymorphic type, false otherwise.

### 6.13.51 IsReferenceType<T0>() Function

Returns true if T0 is an lvalue or rvalue reference type, false otherwise.

**Syntax**

```
public bool IsReferenceType<T0>();
```

## Returns

bool

Returns true if T0 is an lvalue or rvalue reference type, false otherwise.

### 6.13.52 IsRvalueReferenceType<T0>() Function

Returns true if T0 is an rvalue reference type, false otherwise.

#### Syntax

```
public bool IsRvalueReferenceType<T0>();
```

#### Returns

bool

Returns true if T0 is an rvalue reference type, false otherwise.

### 6.13.53 IsSByteType<T0>() Function

Returns true if T0 is sbyte, false otherwise.

#### Syntax

```
public bool IsSByteType<T0>();
```

#### Returns

bool

Returns true if T0 is sbyte, false otherwise.

### 6.13.54 IsShortType<T0>() Function

Returns true if T0 is short, false otherwise.

#### Syntax

```
public bool IsShortType<T0>();
```

#### Returns

bool

Returns true if T0 is short, false otherwise.

### 6.13.55 IsSignedType<T0>() Function

Returns true if T0 is sbyte, short, int, long, false otherwise.

**Syntax**

```
public bool IsSignedType<T0>();
```

## Returns

bool

Returns true if T0 is sbyte, short, int, long, false otherwise.

### 6.13.56 IsUCharType<T0>() Function

Returns true if T0 is uchar, false otherwise.

#### Syntax

```
public bool IsUCharType<T0>();
```

#### Returns

bool

Returns true if T0 is uchar, false otherwise.

### 6.13.57 IsUIntType<T0>() Function

Returns true if T0 is uint, false otherwise.

#### Syntax

```
public bool IsUIntType<T0>();
```

#### Returns

bool

Returns true if T0 is uint, false otherwise.

### 6.13.58 IsULongType<T0>() Function

Returns true if T0 is ulong, false otherwise.

#### Syntax

```
public bool IsULongType<T0>();
```

#### Returns

bool

Returns true if T0 is ulong, false otherwise.

### 6.13.59 IsUShortType<T0>() Function

Returns true if T0 is ushort, false otherwise.

#### Syntax

```
public bool IsUShortType<T0>();
```

#### Returns

bool

Returns true if T0 is ushort, false otherwise.

### 6.13.60 IsUnsignedType<T0>() Function

Returns true if T0 is byte, ushort, uint, ulong, false otherwise.

#### Syntax

```
public bool IsUnsignedType<T0>();
```

#### Returns

bool

Returns true if T0 is byte, ushort, uint, ulong, false otherwise.

### 6.13.61 IsVoidType<T0>() Function

Returns true if T0 is void, false otherwise.

#### Syntax

```
public bool IsVoidType<T0>();
```

#### Returns

bool

Returns true if T0 is void, false otherwise.

### 6.13.62 IsWCharType<T0>() Function

Returns true if T0 is wchar, false otherwise.

#### Syntax

```
public bool IsWCharType<T0>();
```

#### Returns

bool

Returns true if T0 is wchar, false otherwise.

# 7 System.Security Namespace

Contains classes and functions for computing cryptographic hash functions.

## 7.14 Classes

| Class | Description |
|---|---|
| Sha1 | Class for computing SHA-1 hash value. |
| Sha256 | Class for computing SHA-256 hash value. |
| Sha512 | Class for computing SHA-512 hash value. |

## 7.14.1 Sha1 Class

Class for computing SHA-1 hash value.

### Syntax

```
public class Sha1;
```

#### 7.14.1.1 Member Functions

| Member Function | Description |
|---|---|
| Sha1() | Default constructor. |
| Sha1(const System.Security.Sha1&) | Copy constructor. |
| operator=(const System.Security.Sha1&) | Copy assignment. |
| Sha1(System.Security.Sha1&&) | Move constructor. |
| operator=(System.Security.Sha1&&) | Move assignment. |
| GetDigest() | Returns computed SHA-1 message digest as a hexadecimal string. |
| Process(byte) | Process one byte. |
| Process(const void*, const void*) | Process bytes in range begin and end. |
| Process(const void*, int) | Process a block of memory. |
| Reset() | Reset SHA-1 computation ready for computing another message. |

### Sha1() Member Function

Default constructor.

### Syntax

```
public Sha1();
```

### Sha1(const System.Security.Sha1&) Member Function

Copy constructor.

#### Syntax

```
public Sha1(const System.Security.Sha1& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Security.Sha1& | Argument to copy. |

### operator=(const System.Security.Sha1&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Security.Sha1& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Security.Sha1& | Argument to assign. |

### Sha1(System.Security.Sha1&&) Member Function

Move constructor.

#### Syntax

```
public Sha1(System.Security.Sha1&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Security.Sha1&& | Argument to move from. |

### operator=(System.Security.Sha1&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Security.Sha1&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Security.Sha1&& | Argument to assign from. |

### GetDigest() Member Function

Returns computed SHA-1 message digest as a hexadecimal string.

### Syntax

```
public System.String<char> GetDigest();
```

### Returns

System.String<char>

Returns computed SHA-1 message digest as a hexadecimal string.

### Process(byte) Member Function

Process one byte.

### Syntax

```
public void Process(byte x);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| x | byte | A byte to process. |

### Process(const void*, const void*) Member Function

Process bytes in range begin and end.

#### Syntax

```
public void Process(const void* begin, const void* end);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | const void* | Pointer to the beginning of bytes to process. |
| end | const void* | Pointer to the end of bytes to process. |

### Process(const void*, int) Member Function

Process a block of memory.

#### Syntax

```
public void Process(const void* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | const void* | Pointer to memory buffer. |
| count | int | Number of bytes to process. |

### Reset() Member Function

Reset SHA-1 computation ready for computing another message.

### Syntax

```
public void Reset();
```

## 7.14.2 Sha256 Class

Class for computing SHA-256 hash value.

### Syntax

```
public class Sha256;
```

#### 7.14.2.1 Member Functions

| Member Function | Description |
|---|---|
| Sha256() | Default constructor. |
| Sha256(const System.Security.Sha256&) | Copy constructor. |
| operator=(const System.Security.Sha256&) | Copy assignment. |
| Sha256(System.Security.Sha256&&) | Move constructor. |
| operator=(System.Security.Sha256&&) | Move assignment. |
| GetDigest() | Returns computed SHA-256 message digest as a hexadecimal string. |
| Process(byte) | Process one byte. |
| Process(const void*, const void*) | Process bytes in range begin and end. |
| Process(const void*, int) | Process a block of memory. |
| Reset() | Reset SHA-256 computation ready for computing another message. |

### Sha256() Member Function

Default constructor.

### Syntax

```
public Sha256();
```

### Sha256(const System.Security.Sha256&) Member Function

Copy constructor.

#### Syntax

```
public Sha256(const System.Security.Sha256& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Security.Sha256& | Argument to copy. |

### operator=(const System.Security.Sha256&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Security.Sha256& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Security.Sha256& | Argument to assign. |

### Sha256(System.Security.Sha256&&) Member Function

Move constructor.

#### Syntax

```
public Sha256(System.Security.Sha256&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Security.Sha256&& | Argument to move from. |

### operator=(System.Security.Sha256&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Security.Sha256&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Security.Sha256&& | Argument to assign from. |

### GetDigest() Member Function

Returns computed SHA-256 message digest as a hexadecimal string.

### Syntax

```
public System.String<char> GetDigest();
```

### Returns

System.String<char>

Returns computed SHA-256 message digest as a hexadecimal string.

### Process(byte) Member Function

Process one byte.

### Syntax

```
public void Process(byte x);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| x | byte | A byte to process. |

### Process(const void*, const void*) Member Function

Process bytes in range begin and end.

#### Syntax

```
public void Process(const void* begin, const void* end);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | const void* | Pointer to the beginning of bytes to process. |
| end | const void* | Pointer to the end of bytes to process. |

### Process(const void*, int) Member Function

Process a block of memory.

#### Syntax

```
public void Process(const void* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | const void* | Pointer to memory buffer. |
| count | int | Number of bytes to process. |

### Reset() Member Function

Reset SHA-256 computation ready for computing another message.

### Syntax

```
public void Reset();
```

## 7.14.3 Sha512 Class

Class for computing SHA-512 hash value.

### Syntax

```
public class Sha512;
```

#### 7.14.3.1 Member Functions

| Member Function | Description |
|---|---|
| Sha512() | Default constructor. |
| Sha512(const System.Security.Sha512&) | Copy constructor. |
| operator=(const System.Security.Sha512&) | Copy assignment. |
| Sha512(System.Security.Sha512&&) | Move constructor. |
| operator=(System.Security.Sha512&&) | Move assignment. |
| GetDigest() | Returns computed SHA-512 message digest as a hexadecimal string. |
| Process(byte) | Process one byte. |
| Process(const void*, const void*) | Process bytes in range begin and end. |
| Process(const void*, int) | Process a block of memory. |
| Reset() | Reset SHA-512 computation ready for computing another message. |

### Sha512() Member Function

Default constructor.

#### Syntax

```
public Sha512();
```

### Sha512(const System.Security.Sha512&) Member Function

Copy constructor.

#### Syntax

```
public Sha512(const System.Security.Sha512& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Security.Sha512& | Argument to copy. |

### operator=(const System.Security.Sha512&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Security.Sha512& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Security.Sha512& | Argument to assign. |

### Sha512(System.Security.Sha512&&) Member Function

Move constructor.

#### Syntax

```
public Sha512(System.Security.Sha512&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Security.Sha512&& | Argument to move from. |

### operator=(System.Security.Sha512&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Security.Sha512&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Security.Sha512&& | Argument to assign from. |

### GetDigest() Member Function

Returns computed SHA-512 message digest as a hexadecimal string.

### Syntax

```
public System.String<char> GetDigest();
```

### Returns

System.String<char>

Returns computed SHA-512 message digest as a hexadecimal string.

### Process(byte) Member Function

Process one byte.

### Syntax

```
public void Process(byte x);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| x | byte | A byte to process. |

### Process(const void*, const void*) Member Function

Process bytes in range begin and end.

#### Syntax

```
public void Process(const void* begin, const void* end);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| begin | const void* | Pointer to the beginning of bytes to process. |
| end | const void* | Pointer to the end of bytes to process. |

### Process(const void*, int) Member Function

Process a block of memory.

#### Syntax

```
public void Process(const void* buf, int count);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| buf | const void* | Pointer to memory buffer. |
| count | int | Number of bytes to process. |

### Reset() Member Function

Reset SHA-512 computation ready for computing another message.

### Syntax

```
public void Reset();
```

## 7.15 Functions

| Function | Description |
|---|---|
| GetSha1FileDigest(const System.String<char>-&) | Returns SHA-1 hash value for a contents of given file as a hexadecimal string. |
| GetSha1MessageDigest(const System.String<-char>&) | Returns SHA-1 hash value for a given string as a hexadecimal string. |
| GetSha256FileDigest(const System.String<-char>&) | Returns SHA-256 hash value for a contents of given file as a hexadecimal string. |
| GetSha256MessageDigest(const System.String<-char>&) | Returns SHA-256 hash value for a given string as a hexadecimal string. |
| GetSha512FileDigest(const System.String<-char>&) | Returns SHA-512 hash value for a contents of given file as a hexadecimal string. |
| GetSha512MessageDigest(const System.String<-char>&) | Returns SHA-512 hash value for a given string. |
| LeftRotate(uint, uint) | Rotates bits of given uint left by given amount. |
| RightRotate(uint, uint) | Rotates bits of given uint right by given amount. |
| RightRotate(ulong, ulong) | Rotates bits of given ulong right by given amount. |

### 7.15.4 GetSha1FileDigest(const System.String<char>&) Function

Returns SHA-1 hash value for a contents of given file as a hexadecimal string.

#### Syntax

```
public System.String<char> GetSha1FileDigest(const System.String<char>& filePath);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| filePath | const System.String<char>& | A file path. |

#### Returns

System.String<char>

Returns SHA-1 hash value for a contents of given file as a hexadecimal string.

### 7.15.5 GetSha1MessageDigest(const System.String<char>&) Function

Returns SHA-1 hash value for a given string as a hexadecimal string.

#### Syntax

```
public System.String<char> GetSha1MessageDigest(const System.String<char>& message);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| message | const System.String<char>& | A string. |

#### Returns

System.String<char>

Returns SHA-1 hash value for a given string as a hexadecimal string.

### 7.15.6 GetSha256FileDigest(const System.String<char>&) Function

Returns SHA-256 hash value for a contents of given file as a hexadecimal string.

#### Syntax

```
public System.String<char> GetSha256FileDigest(const System.String<char>& filePath);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| filePath | const System.String<char>& | A file path. |

#### Returns

System.String<char>

Returns SHA-256 hash value for a given string as a hexadecimal string.

### 7.15.7 GetSha256MessageDigest(const System.String<char>&) Function

Returns SHA-256 hash value for a given string as a hexadecimal string.

#### Syntax

```
public System.String<char> GetSha256MessageDigest(const System.String<char>& message);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| message | const System.String<char>& | A string. |

## Returns

System.String<char>

Returns SHA-256 hash value for a given string as a hexadecimal string.

### 7.15.8 GetSha512FileDigest(const System.String<char>&) Function

Returns SHA-512 hash value for a contents of given file as a hexadecimal string.

#### Syntax

```
public System.String<char> GetSha512FileDigest(const System.String<char>& filePath);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| filePath | const System.String<char>& | A file path. |

#### Returns

System.String<char>

Returns SHA-512 hash value for a contents of given file as a hexadecimal string.

### 7.15.9 GetSha512MessageDigest(const System.String<char>&) Function

Returns SHA-512 hash value for a given string.

#### Syntax

```
public System.String<char> GetSha512MessageDigest(const System.String<char>& message);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| message | const System.String<char>& | A string. |

#### Returns

System.String<char>

Returns SHA-512 hash value for a given string.

### 7.15.10 LeftRotate(uint, uint) Function

Rotates bits of given uint left by given amount.

#### Syntax

```
public uint LeftRotate(uint x, uint n);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| x | uint | Value to rotate. |
| n | uint | Number of bits to rotate. |

#### Returns

uint

Returns rotated value.

### 7.15.11 RightRotate(uint, uint) Function

Rotates bits of given uint right by given amount.

#### Syntax

```
public uint RightRotate(uint x, uint n);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | uint | Value to rotate. |
| n | uint | Number of bits to rotate. |

## Returns

uint

Returns rotated value.

### 7.15.12 RightRotate(ulong, ulong) Function

Rotates bits of given ulong right by given amount.

#### Syntax

```
public ulong RightRotate(ulong x, ulong n);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| x | ulong | Value to rotate. |
| n | ulong | Number of bits to rotate. |

## Returns

ulong

Returns rotated value.

# 8 System.Text Namespace

Contains classes and functions for manipulating text.

## 8.16 Classes

| Class | Description |
|---|---|
| CodeFormatter | A class for generating indented text. |

## 8.16.1 CodeFormatter Class

A class for generating indented text.

### Syntax

```
public class CodeFormatter;
```

#### 8.16.1.1 Member Functions

| Member Function | Description |
|---|---|
| CodeFormatter() | Default constructor. |
| CodeFormatter(System.IO.OutputStream&) | Constructor. Initalizes the the code formatter with the given output stream. |
| CurrentIndent() const | Returns the current indent in characters. |
| DecIndent() | Decreases the indent. |
| IncIndent() | Increases the indent. |
| Indent() const | Returns the indent level. |
| IndentSize() const | Returns the number of characters to indent. |
| Line() const | Returns current line number. |
| SetIndentSize(int) | Sets the number of characters to indent. |
| SetLine(int) | Sets current line number. |
| Write(const System.String<char>&) | If at the beginning of a line, writes a string indented with the current indent. Otherwise writes the given string. |
| WriteLine() | Writes end-of-line character. |
| WriteLine(const System.String<char>&) | Writes given string using System.Text.-CodeFormatter.Write.System.String.const.ref function and then writes end-of-line character. |

### CodeFormatter() Member Function

Default constructor.

#### Syntax

```
public CodeFormatter();
```

### CodeFormatter(System.IO.OutputStream&) Member Function

Constructor. Initalizes the the code formatter with the given output stream.

#### Syntax

```
public CodeFormatter(System.IO.OutputStream& stream_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| stream_ | System.IO.OutputStream& | An output stream. |

### CurrentIndent() const Member Function

Returns the current indent in characters.

#### Syntax

```
public int CurrentIndent() const;
```

#### Returns

int

Returns the current indent in characters.

### DecIndent() Member Function

Decreases the indent.

#### Syntax

```
public void DecIndent();
```

### IncIndent() Member Function

Increases the indent.

### Syntax

```
public void IncIndent();
```

### Indent() const Member Function

Returns the indent level.

### Syntax

```
public int Indent() const;
```

### Returns

int

Returns the indent level.

### IndentSize() const Member Function

Returns the number of characters to indent.

#### Syntax

```
public int IndentSize() const;
```

#### Returns

int

Returns the number of characters to indent.

### Line() const Member Function

Returns current line number.

#### Syntax

```
public int Line() const;
```

#### Returns

int

Returns current line number.

### SetIndentSize(int) Member Function

Sets the number of characters to indent.

#### Syntax

```
public void SetIndentSize(int indentSize_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| indentSize_ | int | The number of characters to indent. |

### SetLine(int) Member Function

Sets current line number.

#### Syntax

```
public void SetLine(int line_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| line_ | int | Line number. |

### Write(const System.String<char>&) Member Function

If at the beginning of a line, writes a string indented with the current indent. Otherwise writes the given string.

#### Syntax

```
public void Write(const System.String<char>& text);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| text | const System.String<char>& | A string to write. |

### WriteLine() Member Function

Writes end-of-line character.

### Syntax

```
public void WriteLine();
```

### WriteLine(const System.String<char>&) Member Function

Writes given string using System.Text.CodeFormatter.Write.System.String.const.ref function and then writes end-of-line character.

#### Syntax

```
public void WriteLine(const System.String<char>& text);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| text | const System.String<char>& | A string to write. |

## 8.17 Functions

| Function | Description |
|---|---|
| CharStr(char) | Returns a string representation of a character. |
| HexEscape(char) | Returns a hexadecimal representation of the given character. |
| HexEscape(uchar) | Returns a hexadecimal representation of the given UTF-32 character. |
| HexEscape(wchar) | Returns a hexadecimal representation of the given UTF-16 character. |
| MakeCharLiteral(char) | Returns a character literal representation of a character. |
| MakeStringLiteral(const System.String<char>&) | Returns a string literal representation of a string. |
| StringStr(const System.String<char>&) | Returns a string where control characters of given string are escaped. |
| Trim(const System.String<char>&) | Returns a string where white space from the beginning and end of given string are removed. |
| TrimAll(const System.String<char>&) | Returns a string where white space from the beginning, middle and end of given string are removed. Replaces occurrences of white space in the middle of the given string with one space character. |

## 8.17.2 CharStr(char) Function

Returns a string representation of a character.

### Syntax

```
public System.String<char> CharStr(char c);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to convert. |

### Returns

System.String<char>

Returns the string representation of the given character.

### Remarks

If the character is one of the C escape characters, returns a string containing the character prefixed with the backslash. Otherwise if the character is printable, returns a string containing the character. Otherwise returns a string containing the hexadecimal escape of the character.

### 8.17.3 HexEscape(char) Function

Returns a hexadecimal representation of the given character.

#### Syntax

```
public System.String<char> HexEscape(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character. |

#### Returns

System.String<char>

Returns a hexadecimal representation of the given character.

### 8.17.4 HexEscape(uchar) Function

Returns a hexadecimal representation of the given UTF-32 character.

#### Syntax

```
public System.String<char> HexEscape(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A character. |

#### Returns

System.String<char>

Returns a hexadecimal representation of the given character.

### 8.17.5 HexEscape(wchar) Function

Returns a hexadecimal representation of the given UTF-16 character.

#### Syntax

```
public System.String<char> HexEscape(wchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | wchar | A character. |

#### Returns

System.String<char>

Returns a hexadecimal representation of the given character.

### 8.17.6 MakeCharLiteral(char) Function

Returns a character literal representation of a character.

#### Syntax

```
public System.String<char> MakeCharLiteral(char c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | A character to convert. |

#### Returns

System.String<char>

Returns a character literal representation of a character.

#### Remarks

Returns the character enclosed in apostrophes and escaped if necessary.

### 8.17.7 MakeStringLiteral(const System.String<char>&) Function

Returns a string literal representation of a string.

#### Syntax

```
public System.String<char> MakeStringLiteral(const System.String<char>& s);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to convert. |

## Returns

System.String<char>

Returns a string literal representation of a string.

## Remarks

Returns the string enclosed in quotes and characters escaped if necessary.

### 8.17.8 StringStr(const System.String<char>&) Function

Returns a string where control characters of given string are escaped.

#### Syntax

```
public System.String<char> StringStr(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string. |

#### Returns

System.String<char>

Returns a string where control characters of given string are escaped.

### 8.17.9 Trim(const System.String<char>&) Function

Returns a string where white space from the beginning and end of given string are removed.

#### Syntax

```
public System.String<char> Trim(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to trim. |

#### Returns

System.String<char>

Returns a string where white space from the beginning and end of given string are removed.

### 8.17.10 TrimAll(const System.String<char>&) Function

Returns a string where white space from the beginning, middle and end of given string are removed. Replaces occurrences of white space in the middle of the given string with one space character.

#### Syntax

```
public System.String<char> TrimAll(const System.String<char>& s);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | const System.String<char>& | A string to trim. |

#### Returns

System.String<char>

Returns a string where white space from the beginning, middle and end of given string are removed.

# 9 System.Threading Namespace

Contains classes and functions for controlling multiple threads of execution.

## 9.18 Concepts

| Concept | Description |
|---|---|
| Lockable<M> | Lockable class contains Lock() and Unlock() member functions. |

### 9.18.1 Lockable<M> Concept

Lockable class contains Lock() and Unlock() member functions.

#### Syntax

```
public concept Lockable<M>;
```

#### Constraints

void M.Lock();

void M.Unlock();

#### Models

Mutex and RecursiveMutex are models of lockable.

## 9.19 Classes

| Class | Description |
|---|---|
| ConditionVariable | Condition variables can be used as a communication mechanism among threads. |
| LockGuard<M> | Helper class for locking and unlocking lockable object (mutex or recursive mutex). |
| Mutex | Mutexes are a synchronization mechanism for controlling threads' access to some data. |
| RecursiveMutex | Mutexes are a synchronization mechanism for controlling threads' access to some data. A recursive mutex allows the calling thread lock the mutex many times recursively. |
| Thread | Represents thread of execution. |
| ThreadingException | Exception class thrown when some thread operation fails. |

## 9.19.2 ConditionVariable Class

Condition variables can be used as a communication mechanism among threads.

### Syntax

```
public class ConditionVariable;
```

#### 9.19.2.1 Member Functions

| Member Function | Description |
|---|---|
| ConditionVariable() | Constructor. Initializes the condition variable. |
| NotifyAll() | Unblock all threads waiting on this condition variable. |
| NotifyOne() | Unblock one thread waiting on this condition variable. |
| Wait(System.Threading.Mutex&) | Wait on a condition variable to become signaled (notified). Before calling this function, the mutex associated with this condition variable must be locked. |
| WaitFor(System.Threading.Mutex&, System.Duration) | Wait for specified duration that the condition variable to become signaled (notified). Before calling this function, the mutex associated with this condition variable must be locked. |
| WaitUntil(System.Threading.Mutex&, System.TimePoint) | Wait until specified time point that the condition variable to become signaled (notified). Before calling this function, the mutex associated with this condition variable must be locked. |
| ~ConditionVariable() | Destructor. Destroys the condition variable. |

### ConditionVariable() Member Function

Constructor. Initializes the condition variable.

#### Syntax

```
public ConditionVariable();
```

### NotifyAll() Member Function

Unblock all threads waiting on this condition variable.

### Syntax

```
public void NotifyAll();
```

### NotifyOne() Member Function

Unblock one thread waiting on this condition variable.

### Syntax

```
public void NotifyOne();
```

### Wait(System.Threading.Mutex&) Member Function

Wait on a condition variable to become signaled (notified). Before calling this function, the mutex associated with this condition variable must be locked.

#### Syntax

```
public void Wait(System.Threading.Mutex& m);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| m | System.Threading.Mutex& | Mutex associated with this condition variable. |

### WaitFor(System.Threading.Mutex&, System.Duration) Member Function

Wait for specified duration that the condition variable to become signaled (notified). Before calling this function, the mutex associated with this condition variable must be locked.

#### Syntax

```
public bool WaitFor(System.Threading.Mutex& m, System.Duration d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| m | System.Threading.Mutex& | Mutex associated with this condition variable. |
| d | System.Duration | Duration to wait. |

#### Returns

bool

Returns true, if specified duration has elapsed without the condition variable to become signaled, false otherwise.

### WaitUntil(System.Threading.Mutex&, System.TimePoint) Member Function

Wait until specified time point that the condition variable to become signaled (notified). Before calling this function, the mutex associated with this condition variable must be locked.

#### Syntax

```
public bool WaitUntil(System.Threading.Mutex& m, System.TimePoint tp);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| m | System.Threading.Mutex& | Mutex associated with this condition variable. |
| tp | System.TimePoint | Due time. |

#### Returns

bool

Returns true, if specified time point has been reached without the condition variable to become signaled, false otherwise.

### ~ConditionVariable() Member Function

Destructor. Destroys the condition variable.

#### Syntax

```
public ~ConditionVariable();
```

## 9.19.3 LockGuard<M> Class

Helper class for locking and unlocking lockable object (mutex or recursive mutex).

### Syntax

```
public class LockGuard<M>;
```

### Constraint

M is Lockable

#### 9.19.3.1 Member Functions

| Member Function | Description |
|---|---|
| LockGuard<M>() | Default constructor. |
| GetLock() | Returns the lockable object. |
| LockGuard<M>(M&) | Locks the lockable object. |
| ~LockGuard<M>() | Unlocks the lockable object. |

### LockGuard<M>() Member Function

Default constructor.

#### Syntax

```
public LockGuard<M>();
```

### GetLock() Member Function

Returns the lockable object.

#### Syntax

```
public M& GetLock();
```

#### Returns

M&

Returns the lockable object.

### LockGuard<M>(M&) Member Function

Locks the lockable object.

#### Syntax

```
public LockGuard<M>(M& m_);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| m_ | M& | A lockable object. |

### ~LockGuard<M>() Member Function

Unlocks the lockable object.

#### Syntax

```
public ~LockGuard<M>();
```

## 9.19.4 Mutex Class

Mutexes are a synchronization mechanism for controlling threads' access to some data.

### Syntax

```
public class Mutex;
```

### 9.19.4.1 Remarks

Basic mutex cannot be locked recursively many times by same thread.

### 9.19.4.2 Member Functions

| Member Function | Description |
|---|---|
| Mutex() | Constructor. Initializes the mutex. |
| Handle() const | Returns pointer to the mutex handle. |
| Lock() | Locks the mutex. |
| TryLock() | If the mutex is currently unlocked, locks the mutex and returns true. Otherwise returns false. |
| Unlock() | Unlocks the mutex. |
| ~Mutex() | Destructor. Destroys the mutex. |

### Mutex() Member Function

Constructor. Initializes the mutex.

### Syntax

```
public Mutex();
```

### Handle() const Member Function

Returns pointer to the mutex handle.

### Syntax

```
public void** Handle() const;
```

### Returns

void**

Returns pointer to the mutex handle.

### Lock() Member Function

Locks the mutex.

### Syntax

```
public void Lock();
```

### TryLock() Member Function

If the mutex is currently unlocked, locks the mutex and returns true. Otherwise returns false.

#### Syntax

```
public bool TryLock();
```

#### Returns

bool

Returns true, if the mutex is currently unlocked, false otherwise.

### Unlock() Member Function

Unlocks the mutex.

### Syntax

```
public void Unlock();
```

### ~Mutex() Member Function

Destructor. Destroys the mutex.

#### Syntax

```
public ~Mutex();
```

## 9.19.5 RecursiveMutex Class

Mutexes are a synchronization mechanism for controlling threads' access to some data. A recursive mutex allows the calling thread lock the mutex many times recursively.

### Syntax

```
public class RecursiveMutex;
```

### Base Class

System.Threading.Mutex

#### 9.19.5.1 Member Functions

| Member Function | Description |
|---|---|
| RecursiveMutex() | Constructor. Initializes the recursive mutex. |
| ~RecursiveMutex() | Destructor. |

### RecursiveMutex() Member Function

Constructor. Initializes the recursive mutex.

### Syntax

```
public RecursiveMutex();
```

### ~RecursiveMutex() Member Function

Destructor.

#### Syntax

```
public ~RecursiveMutex();
```

## 9.19.6 Thread Class

Represents thread of execution.

### Syntax

```
public class Thread;
```

#### 9.19.6.1 Member Functions

| Member Function | Description |
|---|---|
| Thread() | Default constructor. Initializes a thread that is not associated with operating system thread. |
| Thread(System.Threading.Thread&&) | Move constructor. |
| operator=(System.Threading.Thread&&) | Move assignment. |
| Detach() | Detaches the thread. Detached thread cannot be joined. |
| Handle() const | Returns thread handle. |
| Join() | Waits for the thread to terminate. |
| Thread(System.Threading.ThreadFun, void*) | Constructor. Associates the thread with an operating system thread and starts executing the specified start function asynchronously. |
| ~Thread() | If the thread is associated with an operating system thread and it is not joined or detached, calls exit with EXIT_THREADS_NOT_JOINED exit status. |

### Thread() Member Function

Default constructor. Initializes a thread that is not associated with operating system thread.

### Syntax

```
public Thread();
```

### Thread(System.Threading.Thread&&) Member Function

Move constructor.

#### Syntax

```
public Thread(System.Threading.Thread&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Threading.Thread&& | A thread to move from. |

### operator=(System.Threading.Thread&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Threading.Thread&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Threading.Thread&& | A thread to assign from. |

### Detach() Member Function

Detaches the thread. Detached thread cannot be joined.

### Syntax

```
public void Detach();
```

### Handle() const Member Function

Returns thread handle.

#### Syntax

```
public ulong Handle() const;
```

#### Returns

ulong

Returns thread handle.

### Join() Member Function

Waits for the thread to terminate.

### Syntax

```
public void Join();
```

### Thread(System.Threading.ThreadFun, void*) Member Function

Constructor. Associates the thread with an operating system thread and starts executing the specified start function asynchronously.

#### Syntax

```
public Thread(System.Threading.ThreadFun start, void* arg);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| start | System.Threading.ThreadFun | A thread start function. |
| arg | void* | Argument to the thread start function. |

#### ~Thread() Member Function

If the thread is associated with an operating system thread and it is not joined or detached, calls exit with EXIT_THREADS_NOT_JOINED exit status.

#### Syntax

```
public ~Thread();
```

### 9.19.6.2 Nonmember Functions

| Function | Description |
|---|---|
| operator==(const System.Threading.Thread&, const System.Threading.Thread&) | Returns true, if the specified threads are associated with the same operation system thread, false otherwise. |

### operator==(const System.Threading.Thread&, const System.Threading.Thread&) Function

Returns true, if the specified threads are associated with the same operation system thread, false otherwise.

#### Syntax

```
public bool operator==(const System.Threading.Thread& t1, const System.Threading.Thread&
t2);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| t1 | const System.Threading.Thread& | The first thread. |
| t2 | const System.Threading.Thread& | The second thread. |

#### Returns

bool

Returns true, if the specified threads are associated with the same operation system thread, false otherwise.

## 9.19.7 ThreadingException Class

Exception class thrown when some thread operation fails.

### Syntax

```
public class ThreadingException;
```

### Base Class

System.Exception

#### 9.19.7.1 Member Functions

| Member Function | Description |
|---|---|
| ThreadingException() | Default constructor. |
| ThreadingException(const System.Threading.-ThreadingException&) | Copy constructor. |
| operator=(const System.Threading.-ThreadingException&) | Copy assignment. |
| ThreadingException(System.Threading.-ThreadingException&&) | Move constructor. |
| operator=(System.Threading.-ThreadingException&&) | Move assignment. |
| ThreadingException(const System.String<-char>&, const System.String<char>&) | Constructor. Initializes the threading exception with the specified operation description and failure reason. |
| ~ThreadingException() | Destructor. |

### ThreadingException() Member Function

Default constructor.

#### Syntax

```
public ThreadingException();
```

### ThreadingException(const System.Threading.ThreadingException&) Member Function

Copy constructor.

#### Syntax

```
public ThreadingException(const System.Threading.ThreadingException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Threading.ThreadingException& | Argument to copy. |

### operator=(const System.Threading.ThreadingException&) Member Function

Copy assignment.

#### Syntax

```
public void operator=(const System.Threading.ThreadingException& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | const System.Threading.ThreadingException& | Argument to assign. |

### ThreadingException(System.Threading.ThreadingException&&) Member Function

Move constructor.

#### Syntax

```
public ThreadingException(System.Threading.ThreadingException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Threading.ThreadingException&& | Argument to move from. |

### operator=(System.Threading.ThreadingException&&) Member Function

Move assignment.

#### Syntax

```
public void operator=(System.Threading.ThreadingException&& that);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| that | System.Threading.ThreadingException&& | Argument to assign from. |

### ThreadingException(const System.String<char>&, const System.String<char>&) Member Function

Constructor. Initializes the threading exception with the specified operation description and failure reason.

#### Syntax

```
public ThreadingException(const System.String<char>& operation, const System.String<char>&
reason);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| operation | const System.String<char>& | Description of the thread operation. |
| reason | const System.String<char>& | Reason for failure. |

### ~ThreadingException() Member Function

Destructor.

#### Syntax

```
public ~ThreadingException();
```

## 9.20 Functions

| Function | Description |
|---|---|
| HardwareConcurrency() | Returns the number of cores of the computer. |
| SleepFor(System.Duration) | Puts the calling thread to sleep for specified duration. |
| SleepUntil(System.TimePoint) | Puts the calling thread to sleep until specified time point has been reached. |
| ThreadStart(void*) | Implementation detail. |

### 9.20.8 HardwareConcurrency() Function

Returns the number of cores of the computer.

**Syntax**

```
public int HardwareConcurrency();
```

## Returns

int

Returns the number of cores of the computer.

### 9.20.9 SleepFor(System.Duration) Function

Puts the calling thread to sleep for specified duration.

#### Syntax

```
public void SleepFor(System.Duration d);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| d | System.Duration | Duration to sleep. |

### 9.20.10 SleepUntil(System.TimePoint) Function

Puts the calling thread to sleep until specified time point has been reached.

#### Syntax

```
public void SleepUntil(System.TimePoint tp);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| tp | System.TimePoint | Time point to sleep until. |

### 9.20.11 ThreadStart(void*) Function

Implementation detail.

#### Syntax

```
public void ThreadStart(void* arg);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| arg | void* | |

## 9.21 Delegates

| Delegate | Description |
|---|---|
| ThreadFun | A thread start function delegate. |

### 9.21.12 ThreadFun Delegate

A thread start function delegate.

#### Syntax

```
public delegate void ThreadFun(void* arg);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| arg | void* | Argument given to the thread start function. |

## 9.22 Constants

| Constant | Type | Value | Description |
|---|---|---|---|
| EXIT_THREADS_-NOT_JOINED | int | 250 | Program exit status when all threads are not joined or detached. |

# 10 System.Unicode Namespace

Contains conversions between UTF-8, UTF-16 and UTF-32 strings and Unicode character classification functions.

## 10.23 Classes

| Class | Description |
|---|---|

## 10.24 Functions

| Function | Description |
|---|---|
| GetCategory(uchar) | Returns category enumeration value of given Unicode character. |
| GetCategoryName(System.Unicode.Category) | Returns the name for given Unicode category enumeration value. |
| GetCharacterName(uchar) | Returns a descriptive name in english for given Unicode character. |
| IsLetter(uchar) | Returns true if the given Unicode character is categorized as a letter, false otherwise. |
| IsLower(uchar) | Returns true if the given Unicode character is categorized as a lower case letter, false otherwise. |
| IsMark(uchar) | Returns true if the given Unicode character is categorized as a mark, false otherwise. |
| IsNumber(uchar) | Returns true if the given Unicode character is categorized as a number, false otherwise. |
| IsPunctuation(uchar) | Returns true if the given Unicode character is categorized as punctuation, false otherwise. |
| IsSeparator(uchar) | Returns true if the given Unicode character is categorized as separator character, false otherwise. |
| IsSymbol(uchar) | Returns true if the given Unicode character is categorized as symbol, false otherwise. |
| IsUpper(uchar) | Returns true if the given Unicode character is categorized as an upper case letter, false otherwise. |
| ToLower(uchar) | Converts the given character to lower case. |
| ToUpper(uchar) | Converts the given character to upper case. |
| ToUtf16(const System.String<char>&) | Converts the given UTF-8 encoded string to UTF-16 string. |
| ToUtf16(const System.String<uchar>&) | Converts the given UTF-32 encoded string to UTF-16 string. |
| ToUtf32(const System.String<char>&) | Converts the given UTF-8 encoded string to UTF-32 string. |

| | |
|---|---|
| ToUtf32(const System.String<wchar>&) | Converts the given UTF-16 encoded string to UTF-32 string. |
| ToUtf8(const System.String<uchar>&) | Converts the given UTF-32 encoded string to UTF-8 string. |
| ToUtf8(const System.String<wchar>&) | Converts the given UTF-16 encoded string to UTF-8 string. |

### 10.24.1 GetCategory(uchar) Function

Returns category enumeration value of given Unicode character.

#### Syntax

```
public System.Unicode.Category GetCategory(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

System.Unicode.Category

Returns category enumeration value of given Unicode character.

### 10.24.2 GetCategoryName(System.Unicode.Category) Function

Returns the name for given Unicode category enumeration value.

#### Syntax

```
public System.String<char> GetCategoryName(System.Unicode.Category category);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| category | System.Unicode.Category | A Unicode category enumeration value. |

## Returns

System.String<char>

Returns the name for given Unicode category enumeration value.

## Remarks

The names are commonly two letter strings such as Lu for upper case letters.

### 10.24.3 GetCharacterName(uchar) Function

Returns a descriptive name in english for given Unicode character.

#### Syntax

```
public System.String<char> GetCharacterName(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

System.String<char>

Returns a descriptive name in english for given Unicode character.

### 10.24.4 IsLetter(uchar) Function

Returns true if the given Unicode character is categorized as a letter, false otherwise.

#### Syntax

```
public bool IsLetter(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

bool

Returns true if the given Unicode character is categorized as a letter, false otherwise.

#### Remarks

Letters are in Unicode categories Lu, Ll, LC, Lm, Lo and Lt.

### 10.24.5 IsLower(uchar) Function

Returns true if the given Unicode character is categorized as a lower case letter, false otherwise.

#### Syntax

```
public bool IsLower(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

bool

Returns true if the given Unicode character is categorized as a lower case letter, false otherwise.

#### Remarks

Lower case letters are in Unicode categories Ll.

### 10.24.6 IsMark(uchar) Function

Returns true if the given Unicode character is categorized as a mark, false otherwise.

#### Syntax

```
public bool IsMark(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

bool

Returns true if the given Unicode character is categorized as a mark, false otherwise.

#### Remarks

Marks are in Unicode categories Mc, Me and Mn.

### 10.24.7 IsNumber(uchar) Function

Returns true if the given Unicode character is categorized as a number, false otherwise.

#### Syntax

```
public bool IsNumber(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

bool

Returns true if the given Unicode character is categorized as a number, false otherwise.

#### Remarks

Numbers are in Unicode categories Nd, Nl and No.

### 10.24.8 IsPunctuation(uchar) Function

Returns true if the given Unicode character is categorized as punctuation, false otherwise.

#### Syntax

```
public bool IsPunctuation(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

bool

Returns true if the given Unicode character is categorized as punctuation, false otherwise.

#### Remarks

Punctuation characters are in Unicode categories Pc, Pd, Pe, Pf, Pi, Po and Ps.

### 10.24.9 IsSeparator(uchar) Function

Returns true if the given Unicode character is categorized as separator character, false otherwise.

#### Syntax

```
public bool IsSeparator(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

bool

Returns true if the given Unicode character is categorized as separator character, false otherwise.

#### Remarks

Separator characters are in Unicode categories Zl, Zp and Zs.

### 10.24.10 IsSymbol(uchar) Function

Returns true if the given Unicode character is categorized as symbol, false otherwise.

#### Syntax

```
public bool IsSymbol(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

bool

Returns true if the given Unicode character is categorized as symbol, false otherwise.

#### Remarks

Symbols are in Unicode categories Sc, Sk, Sm and So.

### 10.24.11 IsUpper(uchar) Function

Returns true if the given Unicode character is categorized as an upper case letter, false otherwise.

#### Syntax

```
public bool IsUpper(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

bool

Returns true if the given Unicode character is categorized as an upper case letter, false otherwise.

#### Remarks

Upper case letters are in Unicode categories Lu.

### 10.24.12 ToLower(uchar) Function

Converts the given character to lower case.

#### Syntax

```
public uchar ToLower(uchar c);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

#### Returns

uchar

Converts the given character to lower case.

### 10.24.13 ToUpper(uchar) Function

Converts the given character to upper case.

#### Syntax

```
public uchar ToUpper(uchar c);
```

## Parameters

| Name | Type | Description |
|---|---|---|
| c | uchar | A Unicode character. |

## Returns

uchar

Converts the given character to upper case.

### 10.24.14 ToUtf16(const System.String<char>&) Function

Converts the given UTF-8 encoded string to UTF-16 string.

#### Syntax

```
public System.String<wchar> ToUtf16(const System.String<char>& utf8Str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| utf8Str | const System.String<char>& | A UTF-8 encoded string. |

#### Returns

System.String<wchar>

Converts the given UTF-8 encoded string to UTF-16 string.

### 10.24.15 ToUtf16(const System.String<uchar>&) Function

Converts the given UTF-32 encoded string to UTF-16 string.

#### Syntax

```
public System.String<wchar> ToUtf16(const System.String<uchar>& utf32Str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| utf32Str | const System.String<uchar>& | A UTF-32 encoded string. |

#### Returns

System.String<wchar>

Converts the given UTF-32 encoded string to UTF-16 string.

### 10.24.16 ToUtf32(const System.String<char>&) Function

Converts the given UTF-8 encoded string to UTF-32 string.

#### Syntax

```
public System.String<uchar> ToUtf32(const System.String<char>& utf8Str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| utf8Str | const System.String<char>& | A UTF-8 encoded string. |

#### Returns

System.String<uchar>

Converts the given UTF-8 encoded string to UTF-32 string.

### 10.24.17 ToUtf32(const System.String<wchar>&) Function

Converts the given UTF-16 encoded string to UTF-32 string.

#### Syntax

```
public System.String<uchar> ToUtf32(const System.String<wchar>& utf16Str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| utf16Str | const System.String<wchar>& | A UTF-16 encoded string. |

#### Returns

System.String<uchar>

Converts the given UTF-16 encoded string to UTF-32 string.

### 10.24.18 ToUtf8(const System.String<uchar>&) Function

Converts the given UTF-32 encoded string to UTF-8 string.

#### Syntax

```
public System.String<char> ToUtf8(const System.String<uchar>& utf32Str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| utf32Str | const System.String<uchar>& | A UTF-32 string. |

#### Returns

System.String<char>

Converts the given UTF-32 encoded string to UTF-8 string.

### 10.24.19 ToUtf8(const System.String<wchar>&) Function

Converts the given UTF-16 encoded string to UTF-8 string.

#### Syntax

```
public System.String<char> ToUtf8(const System.String<wchar>& utf16Str);
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| utf16Str | const System.String<wchar>& | A UTF-16 string. |

#### Returns

System.String<char>

Converts the given UTF-16 encoded string to UTF-8 string.

## 10.25 Enumerations

| Enumeration | Description |
|---|---|
| Category | Unicode category enumeration. |

### 10.25.19.1 Category Enumeration

Unicode category enumeration.

**Enumeration Constants**

| Constant | Value | Description |
|---|---|---|
| letter | 63 | Categories Lu, Ll, LC, Lm, Lo and Lt. |
| letterCased | 4 | Category LC. |
| letterLower | 2 | Category Ll. |
| letterModifier | 8 | Category Lm. |
| letterOther | 16 | Category Lo. |
| letterTitle | 32 | Category Lt. |
| letterUpper | 1 | Category Lu. |
| mark | 448 | Categories Mc, Me and Mn. |
| markEnclosing | 128 | Category Me. |
| markNonspacing | 256 | Category Mn. |
| markSpacing | 64 | Category Mc |
| none | 0 | No category. |
| number | 3584 | Categories Nd, Nl and No. |
| numberDecimal | 512 | Category Nd. |
| numberLetter | 1024 | Category Nl. |
| numberOther | 2048 | Category No. |
| punctuation | 520192 | Categories Pc, Pd, Pe, Pf, Pi, Po and Ps. |
| punctuationClose | 16384 | Category Pe. |
| punctuationConnector | 4096 | Category Pc. |
| punctuationDash | 8192 | Category Pd. |
| punctuationFinalQuote | 32768 | Category Pf. |
| punctuationInitialQuote | 65536 | Category Pi. |
| punctuationOpen | 262144 | Category Ps. |
| punctuationOther | 131072 | Category Po. |
| separator | 58720256 | Categories Zl, Zp and Zs. |
| separatorLine | 8388608 | Category Zl. |
| separatorParagraph | 16777216 | Category Zp. |
| separatorSpace | 33554432 | Category Zs. |
| symbol | 7864320 | Categories Sc, Sk, Sm and So. |
| symbolCurrency | 524288 | Category Sc. |
| symbolMath | 2097152 | Category Sm. |
| symbolModifier | 1048576 | Category Sk. |
| symbolOther | 4194304 | Category So. |