AETG Project 2

CSCE 5420 Software Engineering

Mark L. Short

Version 2

Table of Contents

Main Page

Author:
Mark L. Short

Date:
February 9, 2015

Course: CSCE 5420 Software Engineering

Instructor: Dr. Renee Bryce

Assignment:

In this assignment, you will extend your homework 1 submission to provide 3-way combinational coverage.

1. For this assignment, you are not required to take strings as input. You may simply give every factor and every level a unique numerical identifier.
2. You should prompt the user for the number of factors and levels so that the grader can easily grade your program.
3. You may implement the program in your choice of programming language.
4. Your program may end up using too much time or memory for larger inputs, so you will need to be careful in your implementation. The papers do not describe the exact implementation details for the exact data structures that they use. It is expected that you can effectively make decisions that lead to a fast and efficient implementation.
5. There is randomization in your algorithm. You will run your program 100 times for each input and report the average size and execution time for each input. You should also use 50 candidates as done in the AETG literature.
6. You will report the results of your algorithm for several inputs listed in the table on the next page. Of course, it is possible that your results will slightly vary from their reported results since there is randomization in the algorithm. However, if your results are off by more than 20% for any inputs larger than 3^4, you have a bug in your program.
7. You should strive to implement an efficient solution. To receive credit for this assignment, your algorithm cannot run for longer than 20 minutes to find a solution.
8. You are only required to implement a solution for up to 3-way coverage.

AETG algorithm overview:

```
Assume that we have a system with k test parameters and that the i-th parameter has li different values.
Assume that we have already selected r test cases. We select the r+1 by first generating M different
candidate test cases and then choosing one that covers the most new pairs. Each candidate test case
is selected by the following greedy algorithm.

1. Choose a parameter f and a value l for f such that that parameter value appears in the greatest
number of uncovered pairs.

2. Let f1 = f. Then choose a random order for the remaining parameters. Then we have an order for all
k parameters f1, ... fk.

3. Assume that values have been selected for parameters f1... fj. For 1 <= i <= j, let the selected
value for fi be called vi. Then choose a value vj+1 for fj+1 as follows. For each possible value v
for fj, find the number of new pairs in the set of pairs { fj+1 = v and fi = vi for 1 <= i <= j }.
Then let vj+1 be one of the values that appeared in the greatest number of new pairs.

Note that in this step, each parameter value is considered only once for inclusion in a candidate test
case. Also, that when choosing a value for parameter fj+1, the possible values are compared with only
the j values already chosen for parameters f1, ..., fj.
```

Cite:

- RC Bryce and CJ Colbourn, "A density-based greedy algorithm for higher strength covering arrays", Softw. Test. Verif. Reliab. 2009
- D. Richard Kuhn, Raghu N. Kacker, Yu Lei. "Practical Combinatorical Testing" National Institute for Standards and Technology, NIST Special Publication 800-142, October 2010, http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf
- DM Cohen, SR Dalal, ML Fredman, and GC Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design", Appeared in IEEE Transactions On Software Engineering, volume 23, number 7, July 1997, pages 437-444, http://aetgweb.appcomsci.com/papers/1997-tse.html#heur
- Manchester, Bryce, Sampath, Samant, Kuhn, Kacker, "Applying higher strength combinatorial criteria to test case prioritization: a case study", July 27, 2012
- Grindal, Offutt, Andler, "Combination Testing Strategies: A Survey", November 16th, 2004, http://csrc.nist.gov/acts/grindal-offutt-andler.pdf
- George, "Constructing Covering Arrays", Universitat Politecnica De Valencia, https://riunet.upv.es/bitstream/handle/10251/17027/tesisUPV3917.pdf?sequence=1
- Lavavej, "rand() Considered Harmful", Microsoft Corp, http://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful
- Kuhn, Kacker, Yu, Lei, "Introduction to Combinatorial Testing", 2010
- James Bach, Patrick Schroeder, "Pairwise Testing: A Best Practice That Isn't", http://www.satisfice.com
- Ellims, Ince, Petre, "AETG vs. Man: an Assessment of the Effectiveness of Combinatorial Test Data Generation", 2008, Technical Report 2007/08, ISSN 1744-1986
- Bryce, Colbourn, "One-Test-at-a-Time Heuristic Search for Interaction Test Suites", GECCO'07, July 7-11, 2007
- Bryce, Colbourn, "Test Prioritization for Pairwise Interaction Coverage", A-MOST'05, May 15-21, 2005

Implementation Details:

1. The overall greatest challenge in extending the previous assignment of AETG 2-way coverage to 3-way coverage was in the implementation of a systematic way to generate the set of uncovered (UC) 3-way tuples. Ultimately, this was handled through a combinatorial algorithm.
2. Since the tuple is being stored in an unordered set (or a hash-set), with the values of the tuple being used in the hashing algorithm, the nature of extending the tuple from 2 elements to 3 elements required some tweaking & fine tuning to the algorithm. This was accomplished by equal parts research and experimentation.
3. Another fundamental change from the previous assignment was in moving the target platform from x32 to x64.
4. There was extensive amount of time put into testing and performance optimization with various tests performed on a much larger set of AETG configurations than as prescribed in the assignment. Unfortunately, the generated coverage arrays for those configurations exceeded the limits of the **ThreeWay.jar** , which emitted an out-of-range exceptions beginning at test case (v = 6, t = 3, k = 10).
5. At some point, I believe a bug was introduced; however, it did not manifest itself in the test criteria given for the project, but in values used and tested against that were beyond the original scope of the assignment.
6. I have included 3 separate snapshots of test data taken and recorded. I have tagged the entries in Red that are currently suspect.

Snapshot 1 (oldest)

Input	Your best result	Your average result	Your worst result	Average execution time	mAETG
3^{13}	70	71	74	0.3952s	88
2^{10}	16	16	18	0.0426s	18
2^{12}	18	19	21	0.1256s	21
3^{4}	27	31	35	0.0059s	27
6^{6}	334	338	345	0.8762s	343
6^{3}	40	44	47	0.0243s	47
6^{4}	94	104	108	0.0941s	105
6^{10}	1480	1490	1500	32.1978s	1508
7^{5}					229
3^{6}	40	44	48	0.0238s	38
4^{6}	87	104	108	0.0909s	77
10^{6}					1473
5^{6}	195	199	205	0.2983s	194

Snapshot 2 (Bug Introduced)

Input	Your best result	Your average result	Your worst result	Average execution time	mAETG
3^{13}	70	72	78	0.3654s	88
2^{10}	16	17	20	0.0248s	18
2^{12}	18	19	25	0.0498s	21
3^{4}	28	32	39	0.0046s	27
6^{6}	334	342	355	0.1945s	343
6^{3}					47
6^{4}					105
6^{10}	482	488	500	2.4834s	1508
7^{5}					229
3^{6}					38

4^6					77
10^6					1473
5^6	194	201	208	0.0971s	194

Snapshot 3 (most current)

Input	Your best result	Your average result	Your worst result	Average execution time	mAETG
3^{13}	69	71	75	0.4159s	88
2^{10}	16	16	18	0.0277s	18
2^{12}	18	18	20	0.0557s	21
3^4	28	31	35	0.0052s	27
6^6	333	338	346	0.2077s	343
6^3					47
6^4	248	254	262	0.0470s	105
6^{10}	474	480	485	2.8085s	1508
7^5					229
3^6	39	44	49	0.0173s	38
4^6	99	104	109	0.0475s	77
10^6	1479	1488	1498	1.9210s	1473
5^6	194	199	204	0.1085s	194
4^9	135	137	142	0.2853s	125

Todo List

Member CComponentSystem::CalcNumberOfVariableCombinations (WORD nT) const

following uses 20020 bytes of stack space

Class CTestSuite

need to refactor CTestSuite into a class hierarchy

globalScope> Member g_mt

refactor this

globalScope> Member NUM_REPETITIONS

make the following configurable Global const specifying the number of repetitions

globalScope> Member T_WAY

make this configurable

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

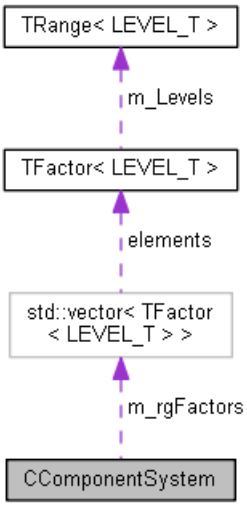
Here is a list of all documented files with brief descriptions:

Class Documentation

CComponentSystem Class Reference

Facilitates the management of the underlying component system.
`#include <ComponentSystem.h>`

Collaboration diagram for CComponentSystem:



Public Member Functions

- CComponentSystem ()
  Default Constructor.
- ~CComponentSystem ()
  Destructor.
- bool Init (FACTOR_T nFactors, LEVEL_T nLevels)
  Class initializer
- bool SetLevelRange (FACTOR_T nFactor, LEVEL_T nMinLevel, LEVEL_T nMaxLevel) throw ()
  Sets the level range for a particular factor.
- LEVEL_T GetMinLevel (FACTOR_T nFactor) const throw ()
  Returns the minimum inclusive level value assigned to a particular factor.
- LEVEL_T GetMaxLevel (FACTOR_T nFactor) const throw ()
  Returns the maximum inclusive level value assigned to a particular factor.
- LEVEL_T GetMaxSystemLevel (void) const throw ()
  Returns the overall maximum inclusive level value assigned to any factor.
- LEVEL_T GetRandomLevel (FACTOR_T nFactor) const throw ()
  Returns a random level value given a target factor.
- FACTOR_T GetFactor (LEVEL_T nLevel) const throw ()
  Returns the factor associated with a given level.
- FACTOR_T GetRandomFactor (void) const throw ()

Returns a random factor value.

- size_t GetShuffledFactors (std::vector< FACTOR_T > &vShuffledFactors) const
  Performs an in-place shuffling of factors.
- size_t CalcNumberOfVariableCombinations (WORD nT) const
  Returns the number of T-way system variable combinations.
- size_t CalcNumberOfTWayConfigurations (WORD nT) const
  Returns the number of T-way variable-value configurations.
- FACTOR_T get_NumFactors (void) const throw ()
  Returns the number of factors currently configured in the system.

Private Attributes

- FACTOR_T m_nNumFactors
  number of factors
- LEVEL_T m_nNumLevels
  number of levels configured per factor
- std::vector< TFactor< LEVEL_T > > m_rgFactors
  array of TFactors

Detailed Description

Facilitates the management of the underlying component system.

In essence, the CComponentSystem class models the valid variable-value configurations as described below:

Definition. For a set of t variables, a variable-value configuration is a set of t valid values for the variables.

Example. Given four binary variables, a, b, c, and d, a=0, c=1, d=0 is a variable-value configuration, and a=1, c=1, d=0 is a different variable- value configuration for the same three variables a, c, and d.

http://csrc.nist.gov/groups/SNS/acts/coverage_measure.html

Member Function Documentation

bool CComponentSystem::Init (FACTOR_T *nFactors*, LEVEL_T *nLevels*)

Class initializer

Parameters:

in	*nFactors*	the max number of system factors permitted
in	*nLevels*	the max number of associated levels per factor permitted

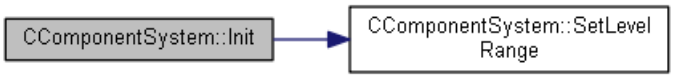
Return values:

true	on success and the system was successfully initialized
false	on error or invalid parameter values

```
25 {
26     bool bResult = false;
27
```

```
28      g_mt.seed(g_rd()); // Initialize the Mersenne Twister engine
29
30      if (( nNumFactors > 0 ) && ( nNumLevels > 0 ))
31      {
32          m_nNumFactors = nNumFactors;
33          m_nNumLevels  = nNumLevels;
34          m_rgFactors.resize(m_nNumFactors);
35
36          for (int nCurFactor = 0; nCurFactor < nNumFactors; nCurFactor++)
37          {
38              LEVEL_T nMinLevel = static_cast<LEVEL_T>(nCurFactor * nNumLevels);
39              LEVEL_T nMaxLevel = nMinLevel + nNumLevels - 1;
40
41              SetLevelRange(nCurFactor, nMinLevel, nMaxLevel);
42          }
43
44          bResult = true;
45      }
46
47      return bResult;
48
49  };
```

Here is the call graph for this function:



bool CComponentSystem::SetLevelRange (FACTOR_T *nFactor*, LEVEL_T *nMinLevel*, LEVEL_T *nMaxLevel*) throw)

Sets the level range for a particular factor.

Parameters:

in	*nFactor*	target factor
in	*nMinLevel*	minimum level value for this factor
in	*nMaxLevel*	maximum level value for this factor

Return values:

true	on success and the factor's level range successfully set
false	on error, target factor is invalid or out-of-range

```
53  {
54      bool bResult = false;
55      if (nFactor < m_nNumFactors)
56      {
57          m_rgFactors[nFactor].SetLevelRange(nMinLevel, nMaxLevel);
58          bResult = true;
59      }
60      return bResult;
61  }
```

LEVEL_T CComponentSystem::GetMinLevel (FACTOR_T *nFactor*) const throw)

Returns the minimum inclusive level value assigned to a particular factor.

Parameters:

in	*nFactor*	target factor

Return values:

LEVEL_T	on success containing the factor's min level value
LEVEL_INVALID	on error

```
65 {
66     LEVEL_T nResult = LEVEL_INVALID;
67
68     if (nFactor < m_nNumFactors)
69         nResult = m_rgFactors[nFactor].get_MinLevel( );
70
71     return nResult;
72 };
```

LEVEL_T CComponentSystem::GetMaxLevel (FACTOR_T *nFactor*) const throw)

Returns the maximum inclusive level value assigned to a particular factor.

Parameters:

in	*nFactor*	target factor

Return values:

LEVEL_INVALID	on error
LEVEL_T	on success containing the factor's max level value

```
76 {
77     LEVEL_T nResult = LEVEL_INVALID;
78
79     if (nFactor < m_nNumFactors)
80         nResult = m_rgFactors[nFactor].get_MaxLevel( );
81
82     return nResult;
83 };
```

LEVEL_T CComponentSystem::GetMaxSystemLevel (void) const throw)

Returns the overall maximum inclusive level value assigned to any factor.

Return values:

LEVEL_T	on success containing the system's max level value
LEVEL_INVALID	on error

```
87 {
88     LEVEL_T nResult = LEVEL_INVALID;
89
90     if (m_nNumFactors > 0)
91         nResult = m_rgFactors[m_nNumFactors - 1].get_MaxLevel( );
92
93     return nResult;
94 };
```

LEVEL_T CComponentSystem::GetRandomLevel (FACTOR_T *nFactor*) const throw)

Returns a random level value given a target factor.

A random number is generated given the range of a factor's configured [min ... max] level value, using the Mersenne Twister engine.

Parameters:

in	*nFactor*	target factor

Return values:

LEVEL_T	on success containing a random generated level
LEVEL_INVALID	on error

```
  98 {
  99     LEVEL_T nResult = LEVEL_INVALID;
 100
 101     if (nFactor < m_nNumFactors)
 102     {
 103         if (std::_Is_UIntType<LEVEL_T>::value)
 104         {
 105             std::uniform_int_distribution<LEVEL_T>
dist(m_rgFactors[nFactor].get_MinLevel(),
 106
m_rgFactors[nFactor].get_MaxLevel());
 107             nResult = dist(g_mt);
 108         }
 109         else
 110         {
 111             std::uniform_int_distribution<int>
dist(m_rgFactors[nFactor].get_MinLevel(),
 112
m_rgFactors[nFactor].get_MaxLevel());
 113             nResult = dist(g_mt);
 114         }
 115     }
 116
 117     return nResult;
 118 }
```

FACTOR_T CComponentSystem::GetFactor (LEVEL_T *nLevel*) const throw)

Returns the factor associated with a given level.

Parameters:

in	*nLevel*	target level

Return values:

FACTOR_T	on success containing a factor value
FACTOR_INVALID	on error or if target level is invalid

```
 144 {
 145     FACTOR_T nResult = FACTOR_INVALID;
 146
 147     if (m_nNumFactors > 0)
 148     {
 149         int iCtr = 0;
 150         for (auto it = m_rgFactors.begin(); (it != m_rgFactors.end()) &&
 151                                     (IsValidFactor(nResult) == false); ++it)
 152         {
 153             if ( it->IsInRange(nLevel) )
 154             {
 155                 nResult = static_cast<FACTOR_T>(iCtr);
 156             }
 157             iCtr++;
 158         }
 159     }
 160
 161     return nResult;
 162 }
```

Here is the call graph for this function:

FACTOR_T CComponentSystem::GetRandomFactor (void) const throw)

Returns a random factor value.

A random factor is generated, given the current configuration in the range of [0..m_nNumFactors - 1], using the Mersenne Twister engine.

Return values:

FACTOR_T	on success containing a random factor value
FACTOR_INVALID	on error

```
122 {
123     FACTOR_T nResult = FACTOR_INVALID;
124
125     if (m_nNumFactors > 1)
126     {
127         if (std::_Is_UIntType<LEVEL_T>::value)
128         {
129             std::uniform_int_distribution<LEVEL_T> dist(0, m_nNumFactors - 1);
130             nResult = dist(g_mt);
131         }
132         else
133         {
134             std::uniform_int_distribution<int>     dist(0, m_nNumFactors - 1);
135             nResult = dist(g_mt);
136         }
137     }
138
139     return nResult;
140 }
```

size_t CComponentSystem::GetShuffledFactors (std::vector< FACTOR_T > & *vShuffledFactors*) const

Performs an in-place shuffling of factors.

This method initializes an sequence of factors based on the current system configuration as [0 .. m_nNumFactors - 1]. It then performs a random in-place shuffling of the sequence using the Mersenne Twister engine.

Parameters:

out	*vShuffledFactors*	target destination

Return values:

size_t	containing number of factors returned
0	on error

```
166 {
167     size_t nResult = 0;
168
169     if (m_nNumFactors > 1) // need at least 2 factors to shuffle
170     {
171         // initialize the array
172         vShuffledFactors.resize(m_nNumFactors);
173         for (int i = 0; i < m_nNumFactors; i++)
174             vShuffledFactors[i] = i;
175
176         // in-place shuffle the array using the Mersenne Twister engine
177         std::shuffle(vShuffledFactors.begin(), vShuffledFactors.end(), g_mt);
178
179         nResult = vShuffledFactors.size();
180     }
181
182     return nResult;
```

```
183 }
```

size_t CComponentSystem::CalcNumberOfVariableCombinations (WORD nT) const

Returns the number of T-way system variable combinations.

The number of t-way combinations in an array of n variables is C(n,t) = n!/(n-t)!t!, or “n choose t” in combinatorics, the number of ways of taking t out of n things at a time.

See also:

http://csrc.nist.gov/groups/SNS/acts/documents/kuhn-kacker-lei-isse-14-preprint.pdf

In this implementation, m_nNumFactors used for 'n' and nT for ’t’, resulting in a calculation of C(m_nFactors, nT).

Parameters:

in	*nT*	the T-way value

Return values:

size_t	containing the number of combinations
0	on error

Todo:

following uses 20020 bytes of stack space

```
187 {
188     size_t nResult = 0;
189 
191     N_Choose_R<50> nChooseR;
192 
193     nResult = nChooseR(m_nNumFactors, nT);
194 
195     return nResult;
196 }
```

size_t CComponentSystem::CalcNumberOfTWayConfigurations (WORD nT) const

Returns the number of T-way variable-value configurations.

Parameters:

in	*nT*	the T-way value

Return values:

size_t	containing the number of configurations

```
200 {
201     size_t nResult = 0;
202 
203     nResult = static_cast<size_t>(pow(m_nNumLevels, nT) *
204                                   CalcNumberOfVariableCombinations(nT));
205 
206     return nResult;
207 }
```

Here is the call graph for this function:

FACTOR_T CComponentSystem::get_NumFactors (void) const throw) [inline]

Returns the number of factors currently configured in the system.

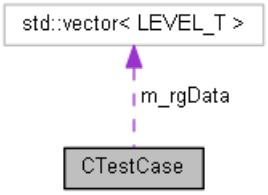
Return values:

FACTOR_T	containing the number of factors

```
346         { return m_nNumFactors; };
```

CTestCase Class Reference

A primitive test case class implementation.
`#include <TestCase.h>`
Collaboration diagram for CTestCase:



Public Types

- typedef LEVEL_VECTOR::reference reference
  exposes associated type definition
- typedef LEVEL_VECTOR::const_reference const_reference
  exposes associated type definition
- typedef LEVEL_VECTOR::const_iterator const_iterator
  exposes associated type definition
- typedef LEVEL_VECTOR::size_type size_type
  exposes associated type definition

Public Member Functions

- CTestCase ()
  Default Constructor.
- CTestCase (const CTestCase &o)
  Copy Constructor.
- CTestCase & operator= (const CTestCase &rhs)
  assignment operator
- void Init (size_t nSize) throw ()
  class initializer
- size_t get_Size (void) const throw ()
  Returns the length of the stored level sequence.
- const_reference operator[] (size_type nPos) const
  subscript to the level nonmutable sequence
- reference operator[] (size_type nPos)
  subscript to the level mutable sequence
- const_iterator begin (void) const throw ()
  Returns a const iterator to the begin of the stored level sequence.
- const_iterator end (void) const throw ()
  Returns a const iterator to the end of the stored level sequence.
- size_t GetNumValidFactors (void) const throw ()
  Returns the number of valid factor entries.
- TCHAR * ToString (TCHAR *szDest, size_t cchLen) const

Writes formatted data to a destination buffer.

Private Types

- typedef std::vector< LEVEL_T > **LEVEL_VECTOR**

Private Attributes

- LEVEL_VECTOR m_rgData
  a collection of level values, indexed by factor

Friends

- std::ostream & operator<< (std::ostream &os, const CTestCase &rhs)
  overloaded stream extraction operator

Detailed Description

A primitive test case class implementation.

The class maintains a collection of (Factor, Level) pairs associated by using a Factor as an index into a sequence of stored Levels.

Member Function Documentation

void CTestCase::Init (size_t *nSize*) throw) [inline]

class initializer

Initializes internal data structures and and allocates storage for variable number of levels to be stored in the test case

Parameters:

in	*nSize*	number of levels to be stored in stored in the CTestCase object

```
80      {
81          // m_rgData.resize(nSize, LEVEL_INVALID);
82           m_rgData.assign(nSize, LEVEL_INVALID);
83      };
```

size_t CTestCase::get_Size (void) const throw) [inline]

Returns the length of the stored level sequence.

Return values:

size_t	containing number of elements

```
91      { return m_rgData.size(); };
```

const_reference CTestCase::operator[] (size_type *nPos*) const [inline]

subscript to the level nonmutable sequence

Parameters:

in	*nPos*	position index

Return values:

const_reference	to nPos'th level in the test case

```
101     { return m_rgData.operator[](nPos); };
```

reference CTestCase::operator[] (size_type *nPos*) [inline]

subscript to the level mutable sequence

Parameters:

in	*nPos*	position index

Return values:

reference	to nPos'th element level in the test case

```
111     { return m_rgData.operator[](nPos); };
```

const_iterator CTestCase::begin (void) const throw) [inline]

Returns a const iterator to the begin of the stored level sequence.

Return values:

const_iterator	to beginning of nonmutable sequence

```
119     { return m_rgData.begin(); };
```

const_iterator CTestCase::end (void) const throw) [inline]

Returns a const iterator to the end of the stored level sequence.

Return values:

const_iterator	to the end of nonmutable sequence

```
127     { return m_rgData.end(); };
```

size_t CTestCase::GetNumValidFactors (void) const throw)

Returns the number of valid factor entries.

Return values:

size_t	contains the number of valid factor elements in the contained level collection

```
 21 {
 22     size_t nResult = 0;
 23     // iterate over our internal data structure
 24     // and count test to if it contains a valid
 25     // level value for each factor entry.
 26     for (auto it : m_rgData)
 27     {
 28         if (IsValidLevel(it))
```

```
29              nResult++;
30      }
31      return nResult;
32  }
```

Here is the call graph for this function:

TCHAR * CTestCase::ToString (TCHAR * *szDest*, size_t *cchLen*) const

Writes formatted data to a destination buffer.

Parameters:

out	*szDest*	storage location for output
in	*cchLen*	count of characters to write

Return values:

*TCHAR**	address of the destination buffer

```
36  {
37      size_t nLen = 0;
38
39      for (auto& it : m_rgData)
40      {
41          nLen += _sntprintf(&szDest[nLen], cchLen - nLen, _T("%d "), it);
42      }
43
44      return szDest;
45  }
```

Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *os*, const CTestCase & *rhs*) [friend]

overloaded stream extraction operator

Parameters:

in,out	*os*	reference to an ostream object
in	*rhs*	target CTestCase object to be written to the stream

Return values:

tostream&	a reference to the resultant stream object

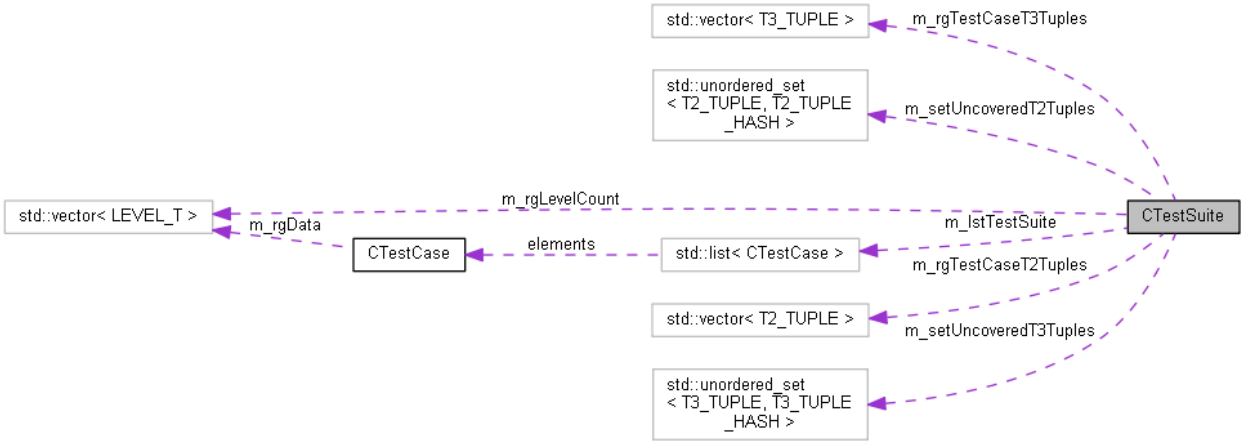
```
12  {
13      for (auto& it : rhs)
14          os << it << _T(" ");
15
16      return os;
17  }
```

CTestSuite Class Reference

Manages test case generation and the collection of test cases.

`#include <TestSuite.h>`

Collaboration diagram for CTestSuite:



Public Types

- typedef std::list< CTestCase >::const_iterator const_iterator
  exposes the underlying typedef

Public Member Functions

- CTestSuite ()
  Default Constructor.
- ~CTestSuite ()
  Default Destructor.
- void Init (const CComponentSystem &system)
  class initializer
- size_t GenerateUncoveredT2Tuples (const CComponentSystem &system)
  Generate the uncovered 2-way tuple set.
- size_t GenerateUncoveredT3Tuples (const CComponentSystem &system)
  Generates the uncovered 3-way tuple set.
- size_t CalculateT2TestCaseCoverage (const CTestCase &TestCase)
  Calculates a coverage value for a given CTestCase object.
- size_t CalculateT3TestCaseCoverage (const CTestCase &TestCase)
  Calculates a coverage value for a given CTestCase object.
- bool GenerateMinLevelCandidate (const CComponentSystem &system, CTestCase &Candidate) const
  Generates a minimum level CTestCase object.
- bool GenerateMaxLevelCandidate (const CComponentSystem &system, CTestCase &Candidate) const
  Generates a maximum level CTestCase object.
- size_t GenerateT2TestCaseCandidate (const CComponentSystem &system, CTestCase &Candidate)
  Generates a 2-way test case candidate.
- size_t GenerateT3TestCaseCandidate (const CComponentSystem &system, CTestCase &Candidate)

Generates a 3-way test case candidate.

- size_t AddToT2TestSuite (const CTestCase &TestCase)
  Adds a new 2-way test case to the suite.
- size_t AddToT3TestSuite (const CTestCase &TestCase)
  Adds a new 3-way test case to the suite.
- size_t get_NumUncoveredT2Tuples (void) const throw ()
  Returns the number of 2-way tuples in the uncovered tuple set.
- size_t get_NumUncoveredT3Tuples (void) const throw ()
  Returns the number of 3-way tuples in the uncovered tuple set.
- size_t get_TestSuiteSize (void) const throw ()
  Returns the current number of CTestCase objects contained in the test suite collection.
- const_iterator begin (void) const throw ()
  Returns an iterator to the beginning of nonmutable sequence of CTestCase objects contained in the current test suite collection.
- const_iterator end (void) const throw ()
  Returns an iterator to the end of nonmutable sequence of CTestCase object contained in the current test suite collection.
- size_t SpliceTestSuite (std::list< CTestCase > &lstOther)
  Transfers ownership of the test suite collection.
- size_t ClearTestSuite (void) throw ()
  Clears the underlying test case collection.

Private Member Functions

- size_t GenerateTestCaseT2Tuples (const CTestCase &TestCase)
  Generates a set of tuples from a given test case.
- size_t GenerateTestCaseT3Tuples (const CTestCase &TestCase)
  Generates a set of tuples from a given test case.

Private Attributes

- FACTOR_T m_nNumFactors
  number of configured system factors
- LEVEL_T m_nMaxSystemLevel
  maximum system level of any factor
- std::vector< LEVEL_T > m_rgLevelCount
  current count of uncovered tuple levels
- std::vector< T2_TUPLE > m_rgTestCaseT2Tuples
  working set of possible T2 tuple coverages
- std::vector< T3_TUPLE > m_rgTestCaseT3Tuples
  working set of possible T3 tuple coverages
- T2_TUPLE_HASHSET m_setUncoveredT2Tuples
  collection of uncovered T2 tuples
- T3_TUPLE_HASHSET m_setUncoveredT3Tuples
  collection of uncovered T3 tuples
- std::list< CTestCase > m_lstTestSuite
  collection of test cases

Detailed Description

Manages test case generation and the and the collection of test cases.

Todo:

need to refactor CTestSuite into a class hierarchy

Member Function Documentation

void CTestSuite::Init (const CComponentSystem & *system*)

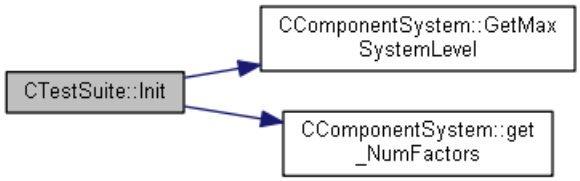
class initializer

Parameters:

in	*system*	

```
38 {
39     m_nMaxSystemLevel = system.GetMaxSystemLevel( );
40     m_nNumFactors     = system.get_NumFactors( );
41
42     m_rgLevelCount.assign(m_nMaxSystemLevel + 1, 0);
43
44     m_setUncoveredT2Tuples.clear();
45     m_setUncoveredT3Tuples.clear();
46     m_lstTestSuite.clear();
47 };
```

Here is the call graph for this function:



size_t CTestSuite::GenerateUncoveredT2Tuples (const CComponentSystem & *system*)

Generate the uncovered 2-way tuple set.

Parameters:

in	*system*	target CComponentSystem

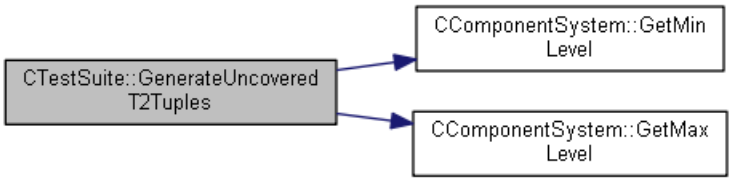
Return values:

size_t	containing the number of uncovered tuples generated

```
51 {
52     m_setUncoveredT2Tuples.clear();
53
54     m_rgLevelCount.assign(m_nMaxSystemLevel + 1, 0);
55
56     for (FACTOR_T nCurFactor = 0; nCurFactor < m_nNumFactors - 1; nCurFactor++)
57     {
58         LEVEL_T nMinFactorLevel = system.GetMinLevel(nCurFactor);
59         LEVEL_T nMaxFactorLevel = system.GetMaxLevel(nCurFactor);
60
```

```
   61         for (LEVEL_T j = nMinFactorLevel; j <= nMaxFactorLevel; j++)
   62         {
   63             for (LEVEL_T k = nMaxFactorLevel + 1; k <= m_nMaxSystemLevel; k++)
   64             {
   65                 auto tuple = std::make_tuple(j, k);
   66
   67                 m_setUncoveredT2Tuples.insert(tuple);
   68                 m_rgLevelCount[j] ++;
   69                 m_rgLevelCount[k] ++;
   70             }
   71         }
   72     }
   73
   74     return m_setUncoveredT2Tuples.size();
   75 }
```

Here is the call graph for this function:



size_t CTestSuite::GenerateUncoveredT3Tuples (const CComponentSystem & *system*)

Generates the uncovered 3-way tuple set.

Parameters:

in	*system*	target CComponentSystem

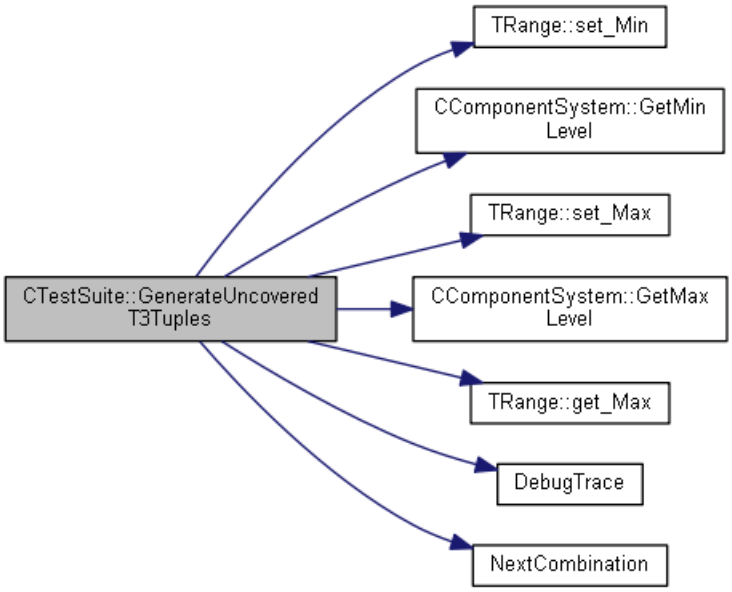
Return values:

size_t	containing the number of uncovered tuples generated

```
   80 {
   81     m_setUncoveredT3Tuples.clear();
   82
   83     m_rgLevelCount.assign(m_nMaxSystemLevel + 1, 0);
   84
   85     std::vector<FACTOR_T> vColumns(T_WAY);
   86
   87     FACTOR_T colFactor = 0;
   88     for (auto& it : vColumns)
   89         it = colFactor++;
   90
   91     bool bMoreCombinations = true;
   92     while( bMoreCombinations )
   93     {
   94         TRange<LEVEL_T> FactorRange[T_WAY];
   95
   96         for (int i = 0; i < T_WAY; i++)
   97         {
   98             FactorRange[i].set_Min(system.GetMinLevel(vColumns[i]));
   99             FactorRange[i].set_Max(system.GetMaxLevel(vColumns[i]));
  100         }
  101
  102         for (LEVEL_T j0 = FactorRange[0].get_Min(); j0 <= FactorRange[0].get_Max(); j0++)
  103         {
  104             for (LEVEL_T j1 = FactorRange[1].get_Min(); j1 <= FactorRange[1].get_Max();
j1++)
  105             {
  106                 for (LEVEL_T j2 = FactorRange[2].get_Min(); j2 <= FactorRange[2].get_Max();
j2++)
```

```
107                 {
108                     auto tuple = std::make_tuple(j0, j1, j2);
109
110                     auto result = m_setUncoveredT3Tuples.insert(tuple);
111 #ifdef _DEBUG
112                     if (result.second == false)
113                         DebugTrace(_T("*** Attempt to insert duplicate tuple *** \n") );
114 #endif
115                     m_rgLevelCount[j0] ++;
116                     m_rgLevelCount[j1] ++;
117                     m_rgLevelCount[j2] ++;
118
119                 }
120             }
121         }
122
123         bMoreCombinations = NextCombination( vColumns.data(), vColumns.size(),
m_nNumFactors);
124     }
125
126     return m_setUncoveredT3Tuples.size();
127 }
```

Here is the call graph for this function:



size_t CTestSuite::CalculateT2TestCaseCoverage (const CTestCase & *TestCase*)

Calculates a coverage value for a given CTestCase object.

Parameters:

in	*TestCase*	target CTestCase object

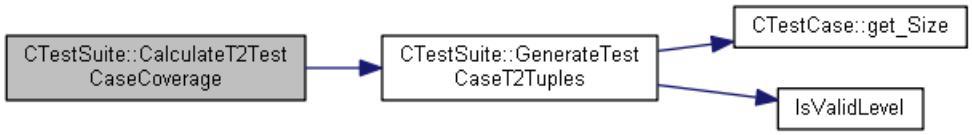
Return values:

size_t	number of 2-way tuples covered in the [partial] test case

```
203 {
204     size_t nResult = 0;
205
206     GenerateTestCaseT2Tuples(TestCase);
207
```

```
208     for (auto& it : m_rgTestCaseT2Tuples)
209     {
210         if (m_setUncoveredT2Tuples.find(it) != m_setUncoveredT2Tuples.end())
211             nResult++;
212     }
213
214     return nResult;
215 }
```

Here is the call graph for this function:



size_t CTestSuite::CalculateT3TestCaseCoverage (const CTestCase & *TestCase*)

Calculates a coverage value for a given CTestCase object.

Parameters:

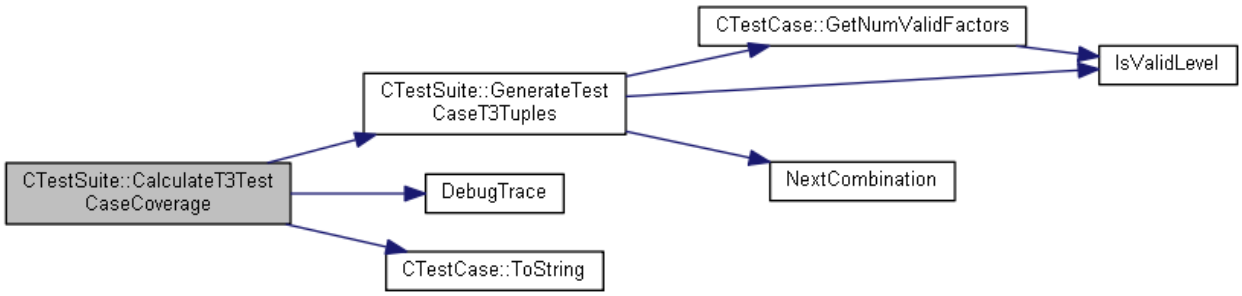
in	*TestCase*	target CTestCase object

Return values:

size_t	number of 3-way tuples covered in the [partial] test case

```
219 {
220     size_t nResult = 0;
221
222     if (GenerateTestCaseT3Tuples(TestCase))
223     {
224         for (auto& it : m_rgTestCaseT3Tuples)
225         {
226             if (m_setUncoveredT3Tuples.find(it) != m_setUncoveredT3Tuples.end( ))
227                 nResult++;
228         }
229     }
230     else
231     {
232 #ifdef _DEBUG
233         TCHAR szBuffer[256] = {0};
234         DebugTrace(_T("** ( %s ) - No Tuples Generated \n"),
235                    TestCase.ToString(szBuffer,  countof(szBuffer) - 1) );
236 #endif
237     }
238
239     return nResult;
240 }
```

Here is the call graph for this function:



bool CTestSuite::GenerateMinLevelCandidate (const CComponentSystem & *system*, CTestCase & *Candidate*) const

Generates a minimum level CTestCase object.

This method generates a test case candidate using the minimum possible level values for all factors given in a system

Parameters:

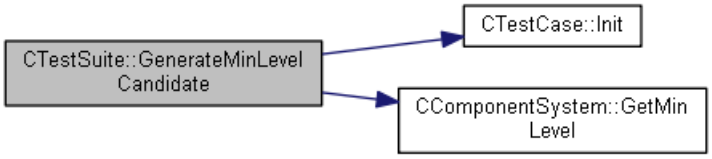
in	*system*	CComponentSystem object, containing the inputs
out	*Candidate*	the resultant CTestCase object

Return values:

true	on success
false	on error

```
588 {
589     bool bResult = false;
590 
591     if (m_nNumFactors > 1)
592     {
593         // initialize the test case candidate
594         Candidate.Init(m_nNumFactors);
595 
596         for (FACTOR_T nCurFactor = 0; nCurFactor < m_nNumFactors; nCurFactor++)
597             Candidate[nCurFactor] = system.GetMinLevel(nCurFactor);
598 
599         bResult = true;
600     }
601 
602     return bResult;
603 };
```

Here is the call graph for this function:



bool CTestSuite::GenerateMaxLevelCandidate (const CComponentSystem & *system*, CTestCase & *Candidate*) const

Generates a maximum level CTestCase object.

This method generates a test case candidate using the maximum possible level values for all factors given in a system

Parameters:

in	*system*	CComponentSystem object, containing the inputs
out	*Candidate*	the resultant CTestCase object

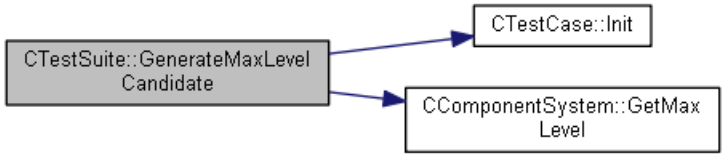
Return values:

true	on success
false	on error

```
608 {
609     bool bResult = false;
610 
611     if (m_nNumFactors > 1)
612     {
```

```
613         // initialize the test case candidate
614         Candidate.Init(m_nNumFactors);
615
616         for (FACTOR_T nCurFactor  = 0; nCurFactor < m_nNumFactors; nCurFactor++)
617             Candidate[nCurFactor] = system.GetMaxLevel(nCurFactor);
618
619         bResult = true;
620     }
621
622     return bResult;
623 };
```

Here is the call graph for this function:



size_t CTestSuite::GenerateT2TestCaseCandidate (const CComponentSystem & *system*, CTestCase & *Candidate*)

Generates a 2-way test case candidate.

Parameters:

in	*system*	CComponentSystem object, containing the inputs
out	*Candidate*	the resultant CTestCase object

Return values:

size_t	containing the candidate coverage value

In AETG, covering arrays are constructed one row at a time. To generate a row, the first t-tuple is selected based on the one involved in most uncovered pairs. Remaining factors are assigned levels in a random order. Levels are selected based on the one that covers the most new t-tuples. For each row that is actually added to the covering array, there are a number, M, candidate rows that are generated and only a candidate that covers the most new t-tuples is added to the covering array. Once a covering is constructed, a number, R, of test suites are generated and the smallest test suite generated is reported. This process continues until all pairs are covered.

AETG Example:

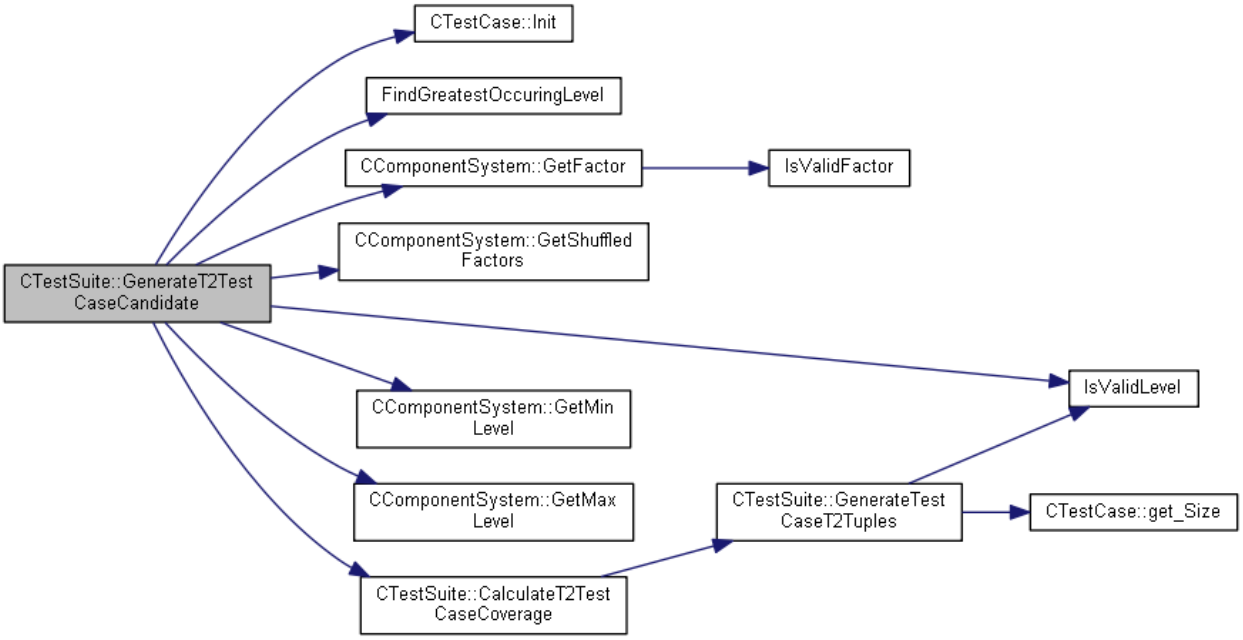
1. initialize test set TS to be an empty set
2. let UC be the set of all t-way combinations to be covered
3. while (UC is not empty) {
4. let p,v be the parameter value that appears the most number of times in UC
5. // generate m candidates
6. for (int i = 0; i < m; i++) {
7. let p be the 1st parameter, and reorder all other parameters randomly, denote the reordered parameters as p1, p2, ... pk.
8. // select values for each candidate
9. for (int j = 1; j <= k; j++) {
10. select a value for the j-th parameter pj such that the most uncovered combinations can be covered
11. }

12. }
13. select a test r from m candidates such that it covers the most uncovered combinations
14. add r into ts and remove from UC the set of combinations covered by r
15. }
16. return ts

```
  395 {
  396     size_t   nResult = 0;
  397
  398     if (m_nNumFactors > 1) // no reason to go further if we don't have at least 2 factors
  399     {
  400         // initialize the candidate
  401         Candidate.Init(m_nNumFactors);
  402
  403         //  1. Choose a parameter f and a value l for f such that that parameter
  404         //     value appears in the greatest number of uncovered pairs.
  405
  406         LEVEL_T  nBestLevel  = FindGreatestOccuringLevel(m_rgLevelCount);
  407         FACTOR_T nCurFactor  = system.GetFactor(nBestLevel);
  408         //   DebugTrace ( T("Greatest Occuring (Factor,Level): (%d, %d) \n"), nFactor,
nLevel);
  409         Candidate[nCurFactor]  = nBestLevel;
  410
  411         //  2. Let f1 = f. Then choose a random order for the remaining parameters.
  412         //     Then we have an order for all k parameters f1, ... fk.
  413
  414         std::vector<FACTOR_T> vFactorOrder;
  415         //  randomize factor order
  416         system.GetShuffledFactors(vFactorOrder);
  417
  418         for (int i = 0; i < vFactorOrder.size( ); i++)
  419         {
  420             nCurFactor = vFactorOrder[i];
  421             // verify to make sure we have not already included it in our candidate
  422             if (IsValidLevel(Candidate[nCurFactor]) == false)
  423             {
  424                 // now we need to determine which of the levels for this factor
  425                 // will cover the greatest number of uncovered t-tuples
  426                 LEVEL_T nMinFactorLevel = system.GetMinLevel(nCurFactor);
  427                 LEVEL_T nMaxFactorLevel = system.GetMaxLevel(nCurFactor);
  428
  429                 nBestLevel = LEVEL_INVALID;
  430                 size_t nCoverage     = 0;
  431                 size_t nBestCoverage = 0;
  432                 int    iRandomMod    = 2;
  433
  434                 for (int j = nMinFactorLevel; j <= nMaxFactorLevel; j++)
  435                 {
  436                     Candidate[nCurFactor] = j;
  437                     nCoverage = CalculateT2TestCaseCoverage(Candidate);
  438
  439                     if (nCoverage > nBestCoverage)
  440                     {
  441                         nBestCoverage = nCoverage;
  442                         nBestLevel    = j;         // identify the level with the best
coverage
  443                         nResult       = nCoverage; // return best coverage value to caller
  444                     }
  445                     else if (nCoverage == nBestCoverage)
  446                     {  // let's randomly determine who is going to be considered the highest
  447                         if (( rand( ) % iRandomMod ) == 0)
  448                         {
  449                             nBestLevel = j;
  450                             iRandomMod++;
  451                         }
  452                     }
  453                 }
  454
  455                 // if we didn't find a level needing to be covered, go
  456                 // ahead and assign something reasonable for the level
  457                 if (IsValidLevel(nBestLevel) == false)
```

```
458                     nBestLevel = nMinFactorLevel;
459
460                 Candidate[nCurFactor] = nBestLevel;
461             }
462         }
463     }
464
465     return nResult;
466 }
```

Here is the call graph for this function:



size_t CTestSuite::GenerateT3TestCaseCandidate (const CComponentSystem & *system*, CTestCase & *Candidate*)

Generates a 3-way test case candidate.

Parameters:

in	*system*	CComponentSystem object, containing the inputs
out	*Candidate*	the resultant CTestCase object

Return values:

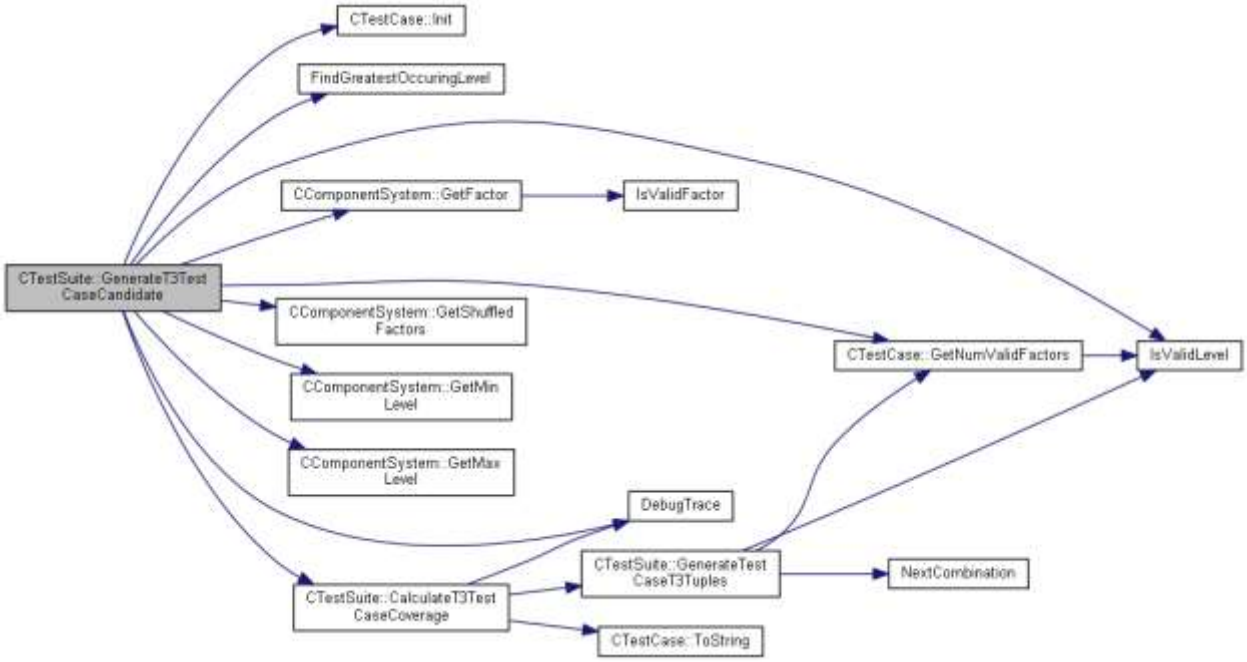
size_t	containing the candidate coverage value

```
470 {
471     size_t   nResult = 0;
472
473     if (m_nNumFactors > 2) // no reason to go further if we don't have at least 3 factors
474     {
475         // initialize the candidate
476         Candidate.Init(m_nNumFactors);
477
478         //  1. Choose a parameter f and a value l for f such that that parameter value appears
479         //     in the greatest number of uncovered pairs.
480
481         LEVEL_T  nBestLevel = FindGreatestOccuringLevel(m_rgLevelCount);
482
```

```
  483 #ifdef _DEBUG
  484         if (IsValidLevel(nBestLevel) == false)
  485             DebugTrace (_T("*** FindGreatestOccuringLevel - LEVEL_INVALID \n") );
  486 #endif
  487         FACTOR_T nCurFactor = system.GetFactor(nBestLevel);
  488
  489 #ifdef _DEBUG
  490 //      DebugTrace ( T("  Greatest Occuring (Factor,Level): (%d, %d) \n"), nCurFactor,
nBestLevel);
  491 #endif
  492         Candidate[nCurFactor] = nBestLevel;
  493
  494         //  2. Let f1 = f. Then choose a random order for the remaining parameters. Then
we have
  495         //     an order for all k parameters f1, ... fk.
  496
  497         std::vector<FACTOR_T> vFactorOrder;
  498         //  randomize factor order
  499         system.GetShuffledFactors(vFactorOrder);
  500
  501         for (int i = 0; i < vFactorOrder.size(); i++)
  502         {
  503             nCurFactor = vFactorOrder[i];
  504             // verify to make sure we have not already included it in our candidate
  505             if (IsValidLevel(Candidate[nCurFactor]) == false)
  506             {
  507                 // now we need to determine which of the levels for this factor
  508                 // will cover the greatest number of uncovered t-tuples
  509                 LEVEL_T nMinFactorLevel = system.GetMinLevel(nCurFactor);
  510                 LEVEL_T nMaxFactorLevel = system.GetMaxLevel(nCurFactor);
  511                 size_t  nCoverage       = 0;
  512                 size_t  nBestCoverage   = 0;
  513                 int     iRandomMod      = 2;
  514                 nBestLevel     = LEVEL_INVALID;
  515                 // Check to see if the Candidate has T_WAY-1 number of valid factors assigned
  516                 if (Candidate.GetNumValidFactors() < T_WAY-1)
  517                 {  // no?
  518                    // then just iterate over the min..max levels for the factor
  519                    // and assign the one that has the highest level count
  520
  521                     for (int n = nMinFactorLevel; n < nMaxFactorLevel; n++)
  522                     {
  523                         if (m_rgLevelCount[n] > nBestCoverage)
  524                         {
  525                             nBestLevel    = n;
  526                             nBestCoverage = m_rgLevelCount[nBestLevel];
  527                         }
  528                         else if ((m_rgLevelCount[n] == nBestCoverage) && (nBestCoverage >
0))
  529                         {// let's randomly determine who is going to be considered the
highest
  530                             if (( rand() % iRandomMod ) == 0)
  531                             {
  532                                 nBestLevel = n;
  533                                 iRandomMod++;
  534                             }
  535                         }
  536                     }
  537                     if (IsValidLevel(nBestLevel) == false)
  538                     {
  539                         std::uniform_int_distribution<LEVEL_T>  dist(nMinFactorLevel,
nMaxFactorLevel);
  540                         nBestLevel = dist(g_mt);
  541                     }
  542
  543                     Candidate[nCurFactor] = nBestLevel;
  544                 }
  545                 else
  546                 {
  547                     for (int j = nMinFactorLevel; j <= nMaxFactorLevel; j++)
  548                     {
```

```
  549                           Candidate[nCurFactor] = j;
  550
  551                           nCoverage = CalculateT3TestCaseCoverage(Candidate);
  552
  553                           if (nCoverage > nBestCoverage)
  554                           {
  555                               nBestCoverage = nCoverage;
  556                               nBestLevel    = j;         // identify the level with the best
coverage
  557                               nResult       = nCoverage; // return best coverage value to
caller
  558                           }
  559                           else if ((nCoverage == nBestCoverage) && (nCoverage > 0))
  560                           {  // let's randomly determine who is going to be considered the
highest
  561                               if (( rand( ) % iRandomMod ) == 0)
  562                               {
  563                                   nBestLevel = j;
  564                                   iRandomMod++;
  565                               }
  566                           }
  567                       }
  568                   }
  569
  570                   // if we didn't find a level needing to be covered, go
  571                   // ahead and assign something reasonable for the level
  572                   if (IsValidLevel(nBestLevel) == false)
  573                   {
  574                       std::uniform_int_distribution<LEVEL_T>  dist(nMinFactorLevel,
nMaxFactorLevel);
  575                       nBestLevel = dist(g_mt);
  576                   }
  577
  578                   Candidate[nCurFactor] = nBestLevel;
  579               }
  580           }
  581       }
  582
  583       return nResult;
  584 }
```

Here is the call graph for this function:



size_t CTestSuite::AddToT2TestSuite (const CTestCase & *TestCase*)

Adds a new 2-way test case to the suite.

Parameters:

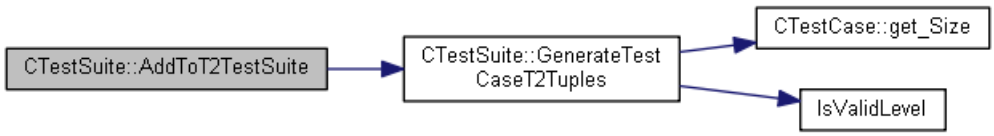
in	*TestCase*	the new CTestCase object to be added

Return values:

size_t	containing the updated number of test cases in the suite after adding the target test case

```
245 {
246     GenerateTestCaseT2Tuples(TestCase);
247 
248     // iterate over the test case tuples and remove them from the
249     // uncovered tuple set
250     for (auto& it : m_rgTestCaseT2Tuples)
251     {
252         if (m_setUncoveredT2Tuples.erase(it) > 0)
253         {
254             int iLvl = std::get<0>(it);
255             m_rgLevelCount[iLvl] --;
256 
257             iLvl = std::get<1>(it);
258             m_rgLevelCount[iLvl] --;
259         }
260     }
261 
262     m_lstTestSuite.push_back(TestCase);
263 
264     return m_lstTestSuite.size();
265 }
```

Here is the call graph for this function:



size_t CTestSuite::AddToT3TestSuite (const CTestCase & *TestCase*)

Adds a new 3-way test case to the suite.

Parameters:

in	*TestCase*	the new CTestCase object to be added

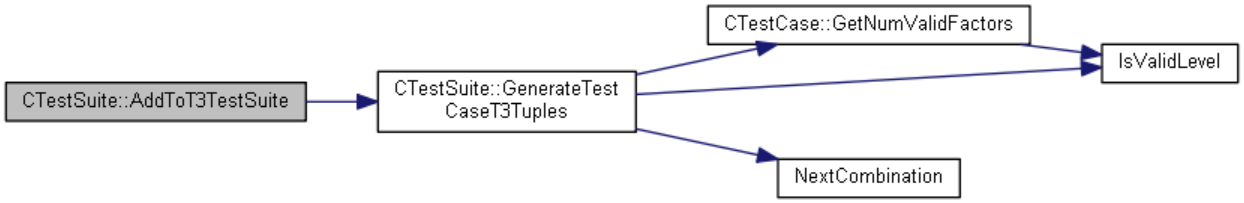
Return values:

size_t	on success, contains the updated number of test cases in the suite after adding the target test case
0	on error

```
269 {
270     size_t nResult = 0;
271 
272     if (GenerateTestCaseT3Tuples(TestCase))
273     {
274         // iterate over the test case tuples and remove them from the
275         // uncovered tuple set
```

```
276         for (auto& it : m_rgTestCaseT3Tuples)
277         {
278             if (m_setUncoveredT3Tuples.erase(it) > 0)
279             {
280                 int iLvl = std::get<0>(it);
281                 m_rgLevelCount[iLvl] --;
282 
283                 iLvl = std::get<1>(it);
284                 m_rgLevelCount[iLvl] --;
285 
286                 iLvl = std::get<2>(it);
287                 m_rgLevelCount[iLvl] --;
288             }
289         }
290 
291         m_lstTestSuite.push_back(TestCase);
292         nResult = m_lstTestSuite.size();
293     }
294 
295     return nResult;
296 }
```

Here is the call graph for this function:



size_t CTestSuite::get_NumUncoveredT2Tuples (void) const throw) [inline]

Returns the number of 2-way tuples in the uncovered tuple set.

Return values:

size_t	containing the current number of uncovered 2-way tuples

```
209     { return m_setUncoveredT2Tuples.size(); };
```

size_t CTestSuite::get_NumUncoveredT3Tuples (void) const throw) [inline]

Returns the number of 3-way tuples in the uncovered tuple set.

Return values:

size_t	containing the current number of uncovered 3-way tuples

```
218     { return m_setUncoveredT3Tuples.size(); };
```

size_t CTestSuite::get_TestSuiteSize (void) const throw) [inline]

Returns the current number of CTestCase objects contained in the test suite collection.

Return values:

size_t	the number of test cases in the test suite

```
228     { return m_lstTestSuite.size(); };
```

const_iterator CTestSuite::begin (void) const throw) [inline]

Returns an iterator to the beginning of nonmutable sequence of CTestCase objects contained in the current test suite collection.

Return values:

const_iterator	pointing to the start of the current test suite

```
239     { return m_lstTestSuite.begin(); };
```

const_iterator CTestSuite::end (void) const throw) [inline]

Returns an iterator to the end of nonmutable sequence of CTestCase object contained in the current test suite collection.

Return values:

const_iterator	pointing to the end of the current test suite

```
249     { return m_lstTestSuite.end(); };
```

size_t CTestSuite::SpliceTestSuite (std::list< CTestCase > & *lstOther*)

Transfers ownership of the test suite collection.

Efficiently detaches and transfers the current collection of CTestCase objects to the target list. No elements are copied or moved, only the internal pointers of the list nodes are re-pointed.

Parameters:

out	*lstOther*	destination list

Return values:

size_t	containing the size of the destination list

```
627 {
628     lstOther.splice(lstOther.begin(), m_lstTestSuite);
629
630     return lstOther.size();
631 }
```

size_t CTestSuite::ClearTestSuite (void) throw)

Clears the underlying test case collection.

Return values:

size_t	containing the size of the test case collection after it has been cleared (which should be 0)

```
635 {
636     m_lstTestSuite.clear();
637     return m_lstTestSuite.size();
638 }
```

size_t CTestSuite::GenerateTestCaseT2Tuples (const CTestCase & *TestCase*) [private]

Generates a set of tuples from a given test case.

Parameters:

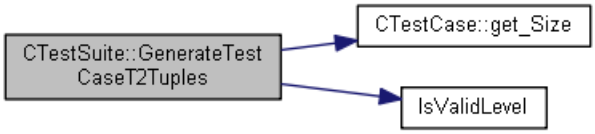
in	*TestCase*	target CTestCase

Return values:

size_t	containing the number of test case 2-way tuples

```
131 {
132     size_t nResult     = 0;
133     size t nNumFactors = TestCase.get Size();
134
135     if (nNumFactors > 1) // need at least 2 factors in the test case
136     {
137         m rgTestCaseT2Tuples.clear();
138         // preallocate enough memory to prevent
139         // incremental memory reallocations as new items are being added
140
141         m rgTestCaseT2Tuples.reserve(nNumFactors * nNumFactors);
142         for (FACTOR T i = 0; i < nNumFactors - 1; i++)
143         {
144             LEVEL T nLevel1 = TestCase[i];
145             if ( IsValidLevel(nLevel1) )
146             {
147                 for (FACTOR T j = i + 1; j <= nNumFactors - 1; j++)
148                 {
149                     LEVEL T nLevel2 = TestCase[j];
150                     if ( IsValidLevel(nLevel2) )
151                     {
152                         auto tuple = std::make tuple(nLevel1, nLevel2);
153                         m rgTestCaseT2Tuples.push_back(tuple);
154                         nResult++;
155                     }
156                 }
157             }
158         }
159     }
160
161     return nResult;
162 }
```

Here is the call graph for this function:



size_t CTestSuite::GenerateTestCaseT3Tuples (const CTestCase & *TestCase*) [private]

Generates a set of tuples from a given test case.

Parameters:

in	*TestCase*	target CTestCase

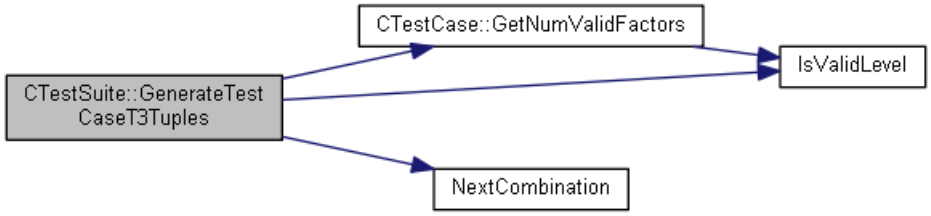
Return values:

size_t	on success, contains the number of test case 3-way tuples

0	on error

```
  166 {
  167     m_rgTestCaseT3Tuples.clear();
  168
  169     if (TestCase.GetNumValidFactors() >= T_WAY) // need at least T_WAY valid factors in the
test case
  170     {
  171         std::vector<FACTOR_T> vColumns(T_WAY);
  172         FACTOR_T colFactor = 0;
  173         for (auto& it : vColumns)
  174             it = colFactor++;
  175
  176         bool bMoreCombinations = true;
  177         while (bMoreCombinations)
  178         {
  179             LEVEL_T nLevel1 = TestCase[vColumns[0]];
  180             if (IsValidLevel(nLevel1))
  181             {
  182                 LEVEL_T nLevel2 = TestCase[vColumns[1]];
  183                 if (IsValidLevel(nLevel2))
  184                 {
  185                     LEVEL_T nLevel3 = TestCase[vColumns[2]];
  186                     if (IsValidLevel(nLevel3))
  187                     {
  188                         auto tuple = std::make_tuple(nLevel1, nLevel2, nLevel3);
  189                         m_rgTestCaseT3Tuples.push_back(tuple);
  190                     }
  191                 }
  192             }
  193             bMoreCombinations = NextCombination(vColumns.data(), vColumns.size(),
m_nNumFactors);
  194 //              bMoreCombinations = NextCombination(vColumns.data(), vColumns.size(),
nNumFactors);
  195         }
  196     }
  197
  198     return m_rgTestCaseT3Tuples.size();
  199 }
```

Here is the call graph for this function:



factorial Struct Reference

`#include <MathUtility.h>`

Public Member Functions

- size_t **operator()** (const size_t n) const

Detailed Description

A very primitive, recursive factorial implementation

N_Choose_R< _Sz > Struct Template Reference

Dynamic programming implementation of C(N,R)
`#include <MathUtility.h>`

Public Member Functions

- size_t **operator()** (const size_t N, const size_t R)

Public Attributes

- size_t **rgData** [_Sz][_Sz]

Detailed Description

template<size_t _Sz>struct N_Choose_R< _Sz >

Dynamic programming implementation of C(N,R)

See also:

http://csg.sph.umich.edu/abecasis/class/2006/615.05.pdf

T2_TUPLE_HASH Struct Reference

A primitive hash function implementation.

`#include <Tuple.h>`

Public Member Functions

- std::size_t operator() (const T2_TUPLE &tpl)
  The underlying hash algorithm implementation

Detailed Description

A primitive hash function implementation.

Performs the necessary hash calculations to allow the T2_TUPLE to be used as a hash key. For compatibility with collections requiring a hash operation, it is important for the resultant key to properly fit within a size_t.

T3_TUPLE_HASH Struct Reference

A primitive hash function implementation.

```
#include <Tuple.h>
```

Public Member Functions

- std::size_t operator() (const T3_TUPLE &tpl)
  the underlying hash algorithm implementation

Detailed Description

A primitive hash function implementation.

Performs the necessary hash calculations to allow the T3_TUPLE to be used as a hash key. For compatibility with collections requiring a hash operation, it is important for the resultant key to properly fit within a size_t.

TFactor< Ty > Class Template Reference

manages ranges associated with a factor

`#include <ComponentSystem.h>`

Public Member Functions

- TFactor ()
  Default Constructor.
- TFactor (Ty nMin, Ty nMax)
  Initialization Constructor.
- void SetLevelRange (Ty nMin, Ty nMax) throw ()
  sets the inclusive level range values
- bool IsInRange (Ty nLevel) const throw ()
  Tests to see if a given level value is within the configured inclusive range values.
- Ty get_MinLevel (void) const throw ()
  Returns the minimum inclusive range value.
- Ty get_MaxLevel (void) const throw ()
  Returns the maximum inclusive range value.

Private Attributes

- TRange< Ty > m_Levels
  associated range of valid levels

Detailed Description

template<class Ty>class TFactor< Ty >

manages ranges associated with a factor

The TFactor class template is a primitive abstraction of a factor and associated permissible level ranges.

Member Function Documentation

template<class Ty> void TFactor< Ty >::SetLevelRange (Ty *nMin*, Ty *nMax*) throw) `[inline]`

Sets the inclusive level range values

Parameters:

in	*nMin*	inclusive minimum range value
in	*nMax*	inclusive maximum range value

```
130     {
131         m_Levels.set_Min(nMin);
132         m_Levels.set_Max(nMax);
133     };
```

template<class Ty> bool TFactor< Ty >::IsInRange (Ty *nLevel*) const throw) [inline]

Tests to see if a given level value is within the configured inclusive range values.

Parameters:

in	*nLevel*	target level value

Return values:

true	if nLevel is within configured level range
false	if nLevel is not within range

```
145     { return m_Levels.IsInRange(nLevel); };
```

template<class Ty> Ty TFactor< Ty >::get_MinLevel (void) const throw) [inline]

Returns the minimum inclusive range value.

Return values:

Ty	containing the minimum inclusive level

```
156     { return m_Levels.get_Min(); };
```

template<class Ty> Ty TFactor< Ty >::get_MaxLevel (void) const throw) [inline]

Returns the maximum inclusive range value.

Return values:

Ty	containing the maximum inclusive level

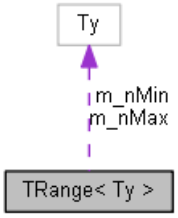
```
164     { return m_Levels.get_Max(); };
```

TRange< Ty > Class Template Reference

a primitive range implementation

`#include <ComponentSystem.h>`

Collaboration diagram for TRange< Ty >:



Public Member Functions

- TRange ()
  Default Constructor.
- TRange (Ty nMin, Ty nMax)
  Initialization Constructor.
- bool IsInRange (Ty nVal) const throw ()
  Performs inclusive range test.
- void set_Min (Ty nSet) throw ()
  Sets the minimum inclusive range.
- void set_Max (Ty nSet) throw ()
  Sets the maximum inclusive range.
- Ty get_Min (void) const throw ()
  Returns the minimum inclusive range value.
- Ty get_Max (void) const throw ()
  Returns the maximum inclusive range value.

Private Attributes

- Ty m_nMin
  inclusive minimum value
- Ty m_nMax
  inclusive maximum value

Detailed Description

template<class Ty>class TRange< Ty >

A primitive range implementation

The TRange template class allows the underlying min & max data types to be determined when the template class is fully declared.

Member Function Documentation

template<class Ty> bool TRange< Ty >::IsInRange (Ty *nVal*) const throw) [inline]

Performs inclusive range test.

Parameters:

in	*nVal*	target value

Return values:

true	if nVal is within range
false	if nVal is out of range

```
62      { return ((m_nMin <= nVal) && (nVal <= m_nMax)); };
```

template<class Ty> void TRange< Ty >::set_Min (Ty *nSet*) throw) [inline]

Sets the minimum inclusive range.

Parameters:

in	*nSet*	minimum range value

```
72      { m_nMin = nSet; };
```

template<class Ty> void TRange< Ty >::set_Max (Ty *nSet*) throw) [inline]

Sets the maximum inclusive range.

Parameters:

in	*nSet*	maximum range value

```
80      { m_nMax = nSet; };
```

template<class Ty> Ty TRange< Ty >::get_Min (void) const throw) [inline]

Returns the minimum inclusive range value.

Return values:

Ty	value that contains the minimum inclusive range

```
88      { return m_nMin; };
```

template<class Ty> Ty TRange< Ty >::get_Max (void) const throw) [inline]

Returns the maximum inclusive range value.

Return values:

Ty	value that contains the maximum inclusive range

```
96      { return m_nMax; };
```

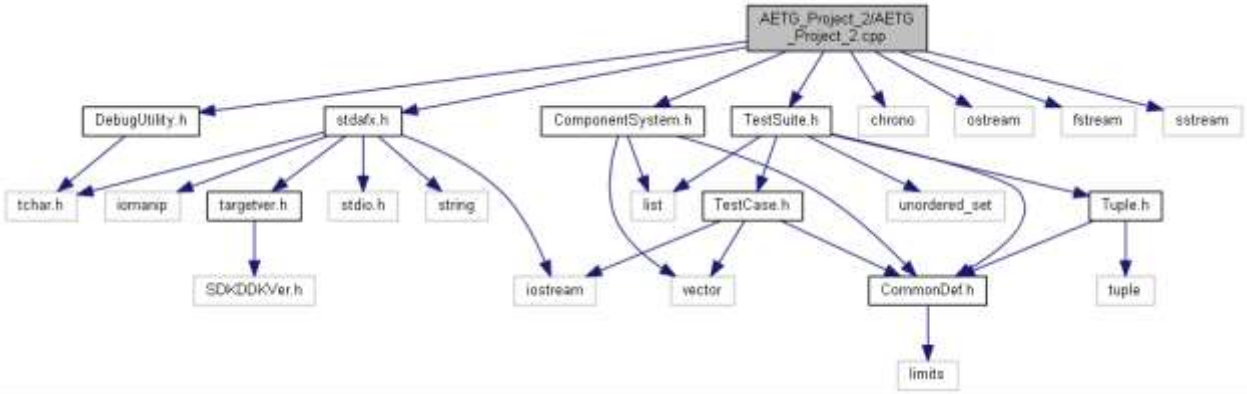
File Documentation

AETG_Project_2/AETG_Project_2.cpp File Reference

Main source file for the AETG_2 project.

```
#include "stdafx.h"
#include "DebugUtility.h"
#include "ComponentSystem.h"
#include "TestSuite.h"
#include <chrono>
#include <ostream>
#include <fstream>
#include <sstream>
```

Include dependency graph for AETG_Project_2.cpp:



Macros

- #define **tostream** std::ostream
- #define **tofstream** std::ofstream
- #define **tstringstream** std::stringstream

Typedefs

- typedef std::chrono::time_point< std::chrono::system_clock > TIME_POINT
  Describes a type that represents a point in time
- typedef std::chrono::duration< double > TIME_DURATION
  Holds a time interval

Functions

- std::ostream & OutputTestSuite (std::ostream &os, const CTestSuite &testSuite)
  Outputs test suite to an ostream.
- std::ostream & OutputTestSuite (std::ostream &os, const std::list< CTestCase > &testSuite)
  Outputs test suite to an ostream.
- int _tmain (int argc, _TCHAR *argv[])
  Main application entry point

Variables

- const int NUM_REPETITIONS = 100
- CComponentSystem g_System
  Global component system object.
- CTestSuite g_TestSuite
  Global test suite object.

Detailed Description

Main source file for the AETG_2 project.

Typedef Documentation

typedef std::chrono::time_point<std::chrono::system_clock> TIME_POINT

Describes a type that represents a point in time

It holds an object of type duration that stores the elapsed time.

typedef std::chrono::duration<double> TIME_DURATION

Holds a time interval

A time interval is an elapsed time between two time points.

Function Documentation

std::ostream& OutputTestSuite (std::ostream & *os*, const CTestSuite & *testSuite*)

Outputs test suite to an ostream.

Parameters:

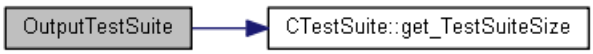
in,out	*os*	reference to an ostream
in	*testSuite*	target

Return values:

tostream	

```
169 {
170     os << testSuite.get TestSuiteSize() << std::endl << std::endl;
171
172     for (auto &it : testSuite )
173         os << it << std::endl;
174
175     return os;
176 }
```

Here is the call graph for this function:



std::ostream& OutputTestSuite (std::ostream & *os*, const std::list< CTestCase > & *testSuite*)

Outputs test suite to an ostream.

Parameters:

in,out	*os*	reference to an ostream
in	*testSuite*	target

Return values:

tostream	

```
187 {
188     os << testSuite.size() << std::endl << std::endl;
189
190     for (auto& it : testSuite)
191         os << it << std::endl;
192
193     return os;
194 }
```

Variable Documentation

const int NUM_REPETITIONS = 100

Todo:

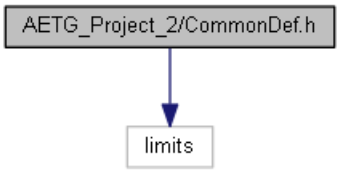
Make the following configurable Global const specifying the number of repetitions

AETG_Project_2/CommonDef.h File Reference

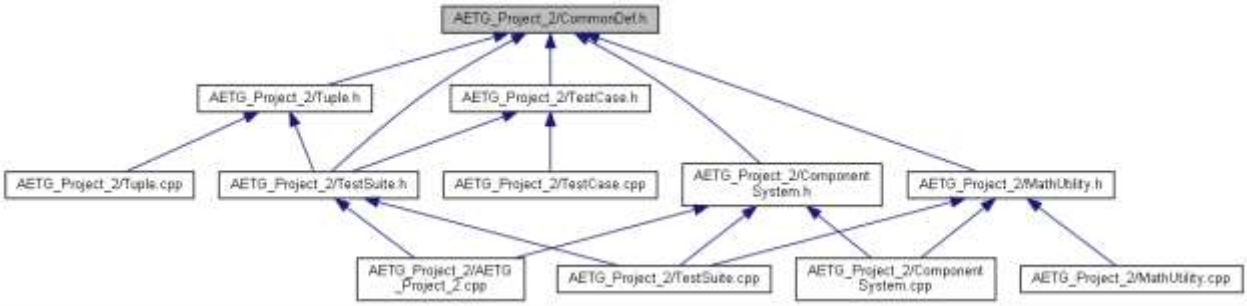
Common type definitions.

```
#include <limits>
```

Include dependency graph for CommonDef.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef unsigned __int8 BYTE
  8-bit unsigned type
- typedef unsigned __int16 WORD
  16-bit unsiged type
- typedef unsigned __int32 DWORD
  32-bit unsigned type
- typedef WORD FACTOR_T
  factor value type
- typedef WORD LEVEL_T
  level value type

Functions

- bool IsValidLevel (LEVEL_T nLevel) throw ()
  Performs basic validation of a level value.
- bool IsValidFactor (FACTOR_T nFactor) throw ()
  Performs basic validation of a factor value.

Variables

- const LEVEL_T LEVEL_INVALID = std::numeric_limits<LEVEL_T>::max()
  used for level validation
- const FACTOR_T FACTOR_INVALID = std::numeric_limits<FACTOR_T>::max()
  used for factor validation

Detailed Description

Common type definitions.

Author:

Mark L. Short

Date:

February 9, 2015

Function Documentation

bool IsValidLevel (LEVEL_T *nLevel*) throw) [inline]

Performs basic validation of a level value.

Parameters:

in	*nLevel*	value to be verified

Return values:

true	if nLevel is valid
false	if nLevel is invalid

```
  40 { return (nLevel != LEVEL_INVALID); };
```

bool IsValidFactor (FACTOR_T *nFactor*) throw) [inline]

Performs basic validation of a factor value.

Parameters:

in	*nFactor*	value to be verified

Return values:

true	if nFactor is valid
false	if nFactor is invalid

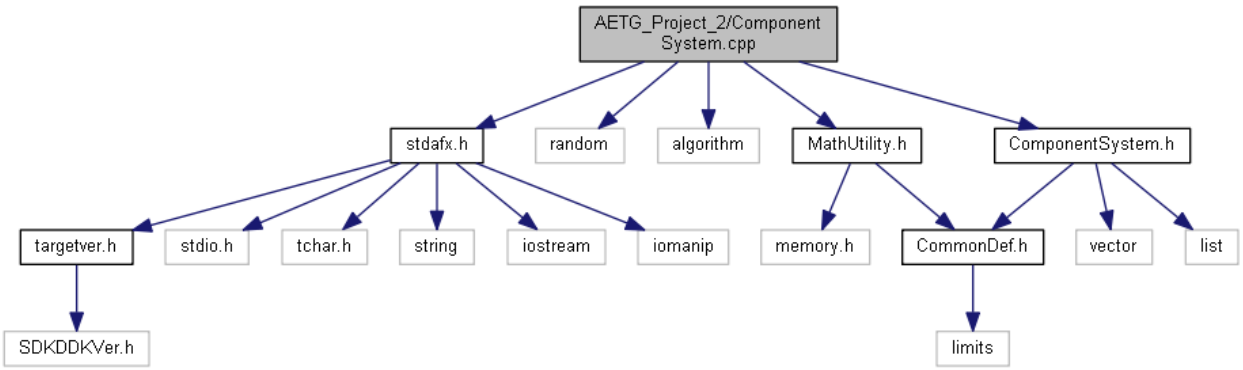
```
  51 { return (nFactor != FACTOR_INVALID); };
```

AETG_Project_2/ComponentSystem.cpp File Reference

CComponentSystem class implementation.

```
#include "stdafx.h"
#include <random>
#include <algorithm>
#include "MathUtility.h"
#include "ComponentSystem.h"
```

Include dependency graph for ComponentSystem.cpp:



Variables

- std::random_device g_rd
  used to generate a 32-bit seed
- std::mt19937 g_mt
  the Mersenne Twister engine

Detailed Description

CComponentSystem class implementation.

Author:
Mark L. Short

Date:
February 9, 2015

Variable Documentation

std::mt19937 g_mt

the Mersenne Twister engine

Todo:

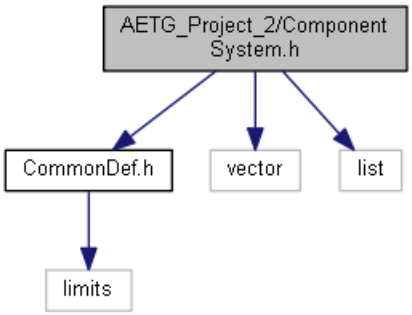
Refactor this

AETG_Project_2/ComponentSystem.h File Reference

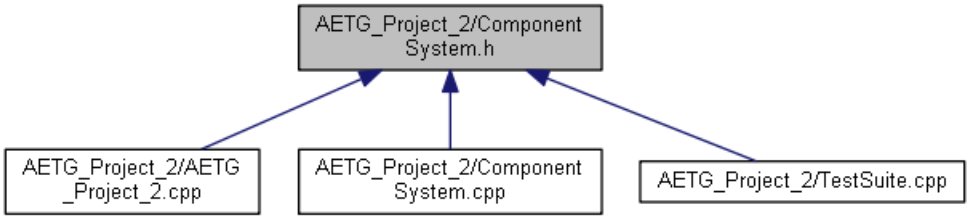
CComponentSystem class interface.

```
#include "CommonDef.h"
#include <vector>
#include <list>
```

Include dependency graph for ComponentSystem.h:



This graph shows which files directly or indirectly include this file:



Classes

- class TRange< Ty >
  a primitive range implementation
- class TFactor< Ty >
  manages ranges associated with a factor
- class CComponentSystem
  Facilitates the management of the underlying component system.

Detailed Description

CComponentSystem class interface.

Provides type definitions for: TRange, TFactor, CComponentSystem

Author:

Mark L. Short

Date:

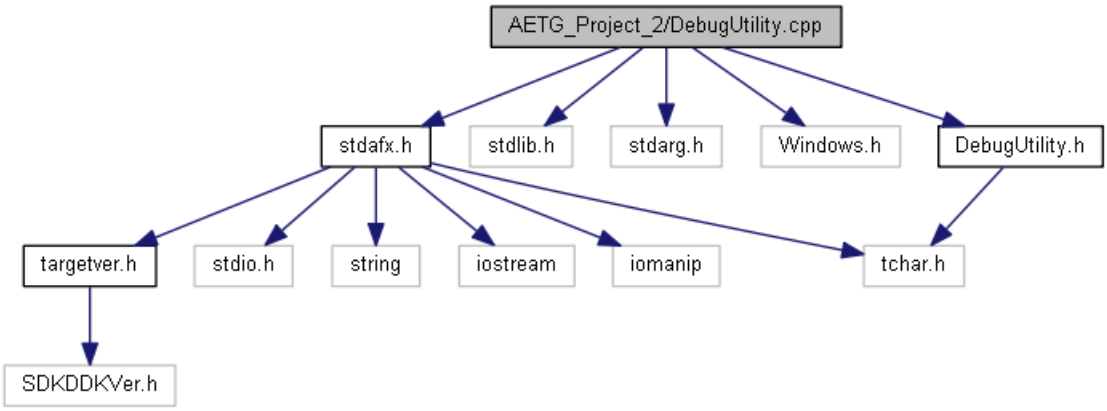
February 9, 2015

AETG_Project_2/DebugUtility.cpp File Reference

Implementation of DebugUtility.cpp.

```
#include "stdafx.h"
#include <stdlib.h>
#include <stdarg.h>
#include <Windows.h>
#include "DebugUtility.h"
```

Include dependency graph for DebugUtility.cpp:



Functions

- int DebugTrace (const TCHAR *szFmt,...)
  Directs output to the IDE output window.
- TCHAR * GetModulePath (TCHAR *szModulePath, size_t cchLen)
  Retrieves the current executable directory.

Detailed Description

Implementation of DebugUtility.cpp.

Author:
Mark L. Short

Date:
February 9, 2015

Function Documentation

int DebugTrace (const TCHAR * *szFmt*, ...)

Directs output to the IDE output window.

Parameters:

in	*szFmt*	printf-styled format string

Return values:

int	the number of characters written if the number of characters to write is less than or equal to count; if the number of characters to write is greater than count, the function returns -1 indicating that output has been truncated. The return value does not include the terminating null, if one is written.

```
   27 {
   28     TCHAR szDebugMsg[512] = { 0 };
   29
   30     va list vaArgs;
   31     va_start (vaArgs, szFmt);
   32
   33     // use the format string and arguments to construct the debug output string
   34     int iReturnVal =  vsntprintf (szDebugMsg,  countof (szDebugMsg) - 1, szFmt, vaArgs);
   35     va end (vaArgs);
   36
   37     ::OutputDebugString (szDebugMsg);
   38     return iReturnVal;
   39
   40 }
```

TCHAR* GetModulePath (TCHAR * *szModulePath*, size_t *cchLen*)

Retrieves the current executable directory.

Parameters:

out	*szModulePath*	destination memory address used to write application's directory path
in	*cchLen*	count of characters available to be written in the destination buffer

Return values:

*TCHAR**	destination address
NULL	on error

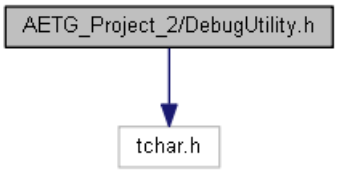
```
   43 {
   44     TCHAR* szReturnVal = NULL;
   45
   46     // Get the executable file path
   47     TCHAR szModuleFileName[ MAX PATH] = { 0 };
   48
   49     // Note, if HANDLE is NULL, GetModuleFileName is supposed to return the file path to
the
   50     // current executable, but it appears that it is inconsistently returning filename as
   51     // well....
   52     DWORD dwStrLen = ::GetModuleFileName (NULL, szModuleFileName,
_countof(szModuleFileName) );
   53
   54     if (dwStrLen != 0)
   55     {
   56         TCHAR szDir[_MAX_PATH] = { 0 };
   57
   58         _tsplitpath(szModuleFileName, szDir, &szDir[_MAX_DRIVE-1], NULL, NULL);
   59
   60         szReturnVal = _tcsncpy(szModulePath, szDir, cchLen);
   61     }
   62
   63     return szReturnVal;
   64 }
```

AETG_Project_2/DebugUtility.h File Reference

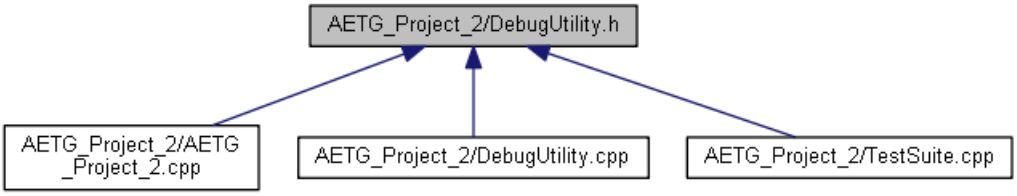
Debugging and utility method declarations.

```
#include <tchar.h>
```

Include dependency graph for DebugUtility.h:



This graph shows which files directly or indirectly include this file:



Functions

- int DebugTrace (const TCHAR *szFmt,...) throw ()
  Directs output to the IDE output window.
- TCHAR * GetModulePath (TCHAR *szModulePath, size_t cchLen) throw ()
  Retrieves the current executable directory.

Detailed Description

Debugging and utility method declarations.

Author:
Mark L. Short

Date:
February 9, 2015

Function Documentation

int DebugTrace (const TCHAR * *szFmt*, ...) throw)

Directs output to the IDE output window.

Parameters:

in	*szFmt*	printf-styled format string

Return values:

int	the number of characters written if the number of characters to write is less than or equal to count; if the number of characters to write is greater than count, the function returns -1 indicating that output has been truncated. The return value does not include the terminating null, if one is written.

```
   27 {
   28     TCHAR szDebugMsg[512] = { 0 };
   29
   30     va_list vaArgs;
   31     va_start (vaArgs, szFmt);
   32
   33     // use the format string and arguments to construct the debug output string
   34     int iReturnVal =  vsntprintf (szDebugMsg,  countof (szDebugMsg) - 1, szFmt, vaArgs);
   35     va_end (vaArgs);
   36
   37     ::OutputDebugString (szDebugMsg);
   38     return iReturnVal;
   39
   40 }
```

TCHAR* GetModulePath (TCHAR * *szModulePath*, size_t *cchLen*) throw)

Retrieves the current executable directory.

Parameters:

out	*szModulePath*	destination memory address used to write application's directory path
in	*cchLen*	count of characters available to be written in the destination buffer

Return values:

*TCHAR**	destination address
NULL	on error

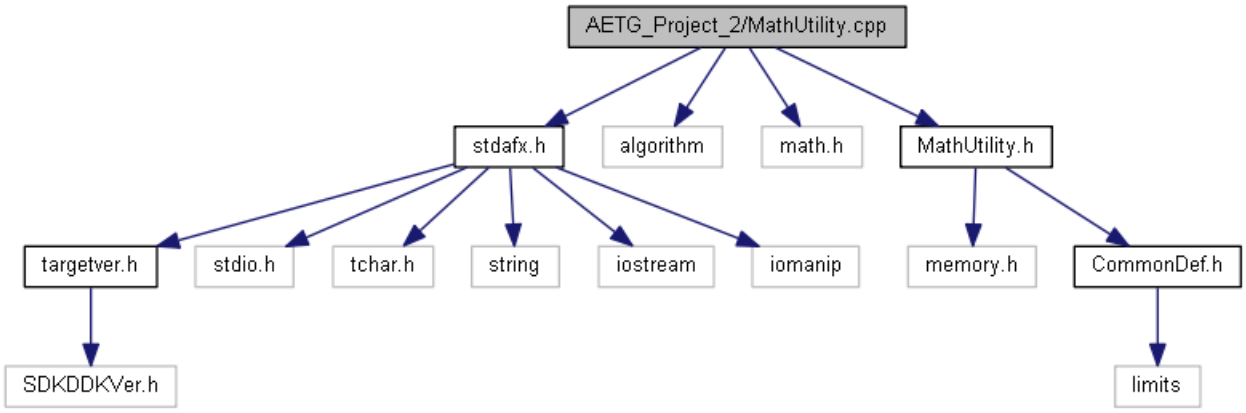
```
   43 {
   44     TCHAR* szReturnVal = NULL;
   45
   46     // Get the executable file path
   47     TCHAR szModuleFileName[ MAX PATH] = { 0 };
   48
   49     // Note, if HANDLE is NULL, GetModuleFileName is supposed to return the file path to
the
   50     // current executable, but it appears that it is inconsistently returning filename as
   51     // well....
   52     DWORD dwStrLen = ::GetModuleFileName (NULL, szModuleFileName,
_countof(szModuleFileName) );
   53
   54     if (dwStrLen != 0)
   55     {
   56         TCHAR szDir[ MAX PATH] = { 0 };
   57
   58         _tsplitpath(szModuleFileName, szDir, &szDir[_MAX_DRIVE-1], NULL, NULL);
   59
   60         szReturnVal = _tcsncpy(szModulePath, szDir, cchLen);
   61     }
   62
   63     return szReturnVal;
   64 }
```

AETG_Project_2/MathUtility.cpp File Reference

Implementation of MathUtility.cpp.

```
#include "stdafx.h"
#include <algorithm>
#include <math.h>
#include "MathUtility.h"
```

Include dependency graph for MathUtility.cpp:



Functions

- size_t **N_Choose_T** (unsigned short nN, unsigned short nT)
- bool NextCombination (unsigned short *rgSubset, size_t nSubset_size, size_t nSet_size)
  Generates a sequence of numbers using a combinatorical operation.
- bool **NextCombination** (BYTE *rgSubset, size_t nSubset_size, size_t nSet_size)

Detailed Description

Implementation of MathUtility.cpp.

Author:
Mark L. Short

Date:
February 9, 2014

Function Documentation

bool NextCombination (unsigned short * *rgSubset*, size_t *nSubset_size*, size_t *nSet_size*)

Generates a sequence of numbers using a combinatorical operation.

The following function generates the next combination (lexicographically) given a current sequence of numbers out of a set of nSetsize

Parameters:

in,out	*rgSubset*	address of a sequence of numbers to be used as the source and destination in generating the next combinatorical sequence
in	*nSubset_size*	the number of elements in rgSubset
in	*nSet_size*	the number of elements in the overall set

Return values:

true	if there are more combinations
false	if there are no more combinations

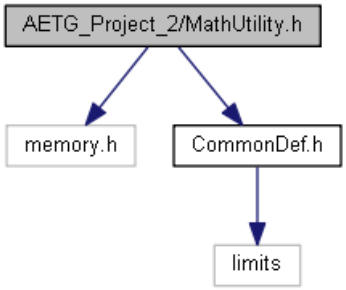
```
 34 {
 35     for (int i = nSubset_size; --i >= 0;)
 36     {
 37         if (++rgSubset[i] <= nSet_size - (nSubset_size - i) )
 38         {
 39             while (++i < nSubset size)
 40                 rgSubset[i] = rgSubset[i-1] + 1;
 41             return true;
 42         }
 43     }
 44     return false;
 45 }
```

AETG_Project_2/MathUtility.h File Reference

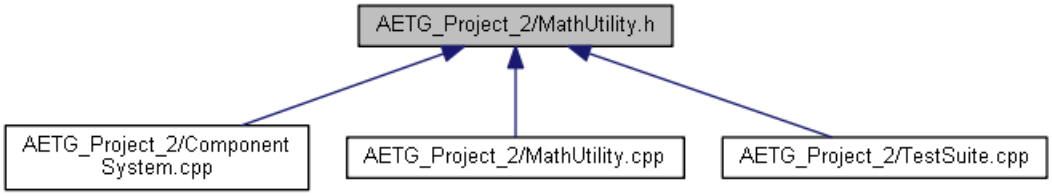
Mathematical utility method declarations.

```
#include <memory.h>
#include "CommonDef.h"
```

Include dependency graph for MathUtility.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct factorial
- struct N_Choose_R< Sz >

Dynamic programming implementation of C(N,R) Functions

- size_t **N_Choose_T** (unsigned short nN, unsigned short nT)
- bool NextCombination (unsigned short *rgSubset, size_t nSubset_size, size_t nSet_size)
  Generates a sequence of numbers using a combinatorial operation.
- bool **NextCombination** (BYTE *rgSubset, size_t nSubset_size, size_t nSet_size)

Detailed Description

Mathematical utility method declarations.

Author:

Mark L. Short

Date:

February 9, 2014

Function Documentation

bool NextCombination (unsigned short * *rgSubset*, size_t *nSubset_size*, size_t *nSet_size*)

Generates a sequence of numbers using a combinatorial operation.

The following function generates the next combination (lexicographically) given a current sequence of numbers out of a set of nSetsize

Parameters:

in,out	*rgSubset*	address of a sequence of numbers to be used as the source and destination in generating the next combinatorial sequence
in	*nSubset_size*	the number of elements in rgSubset
in	*nSet_size*	the number of elements in the overall set

Return values:

true	if there are more combinations
false	if there are no more combinations

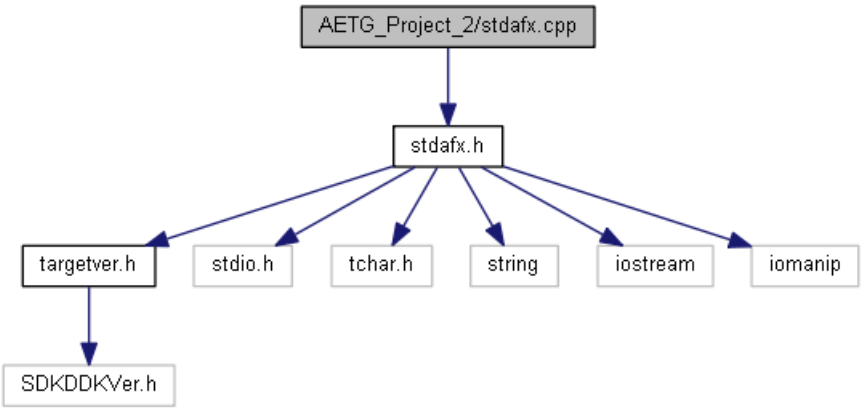
```
34 {
35     for (int i = nSubset_size; --i >= 0;)
36     {
37         if (++rgSubset[i] <= nSet_size - (nSubset_size - i) )
38         {
39             while (++i < nSubset_size)
40                 rgSubset[i] = rgSubset[i-1] + 1;
41             return true;
42         }
43     }
44     return false;
45 }
```

AETG_Project_2/stdafx.cpp File Reference

Source file that includes just the standard headers.

```
#include "stdafx.h"
```

Include dependency graph for stdafx.cpp:



Detailed Description

Source file that includes just the standard headers.

AETG_Project_2.pch will be the pre-compiled header & stdafx.obj will contain the pre-compiled type information.

Author:

Mark L. Short

Date:

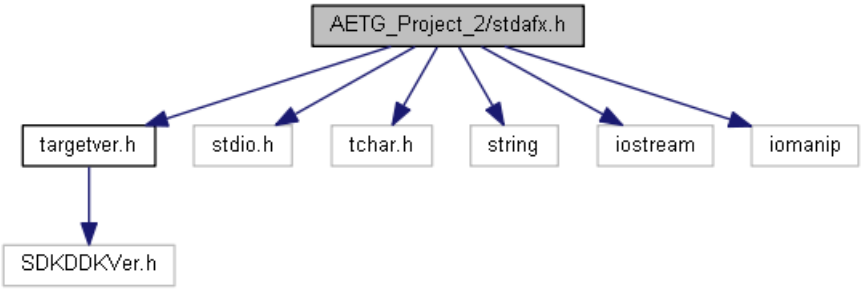
February 9, 2015

AETG_Project_2/stdafx.h File Reference

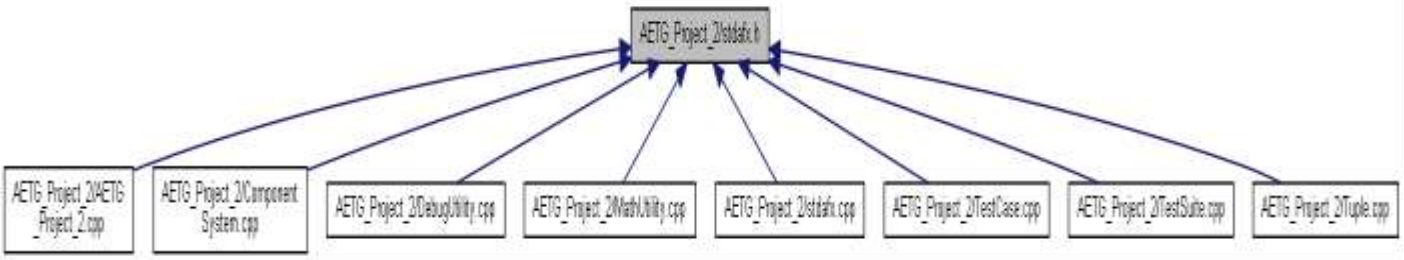
Application header file.

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <string>
#include <iostream>
#include <iomanip>
```

Include dependency graph for stdafx.h:



This graph shows which files directly or indirectly include this file:



Detailed Description

Application header file.

Include file for standard system include header files, or project specific include files that are used frequently, but are changed infrequently

Author:

Mark L. Short

Date:

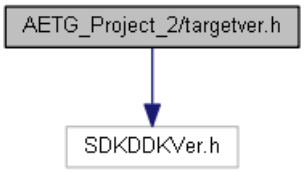
February 9, 2015

AETG_Project_2/targetver.h File Reference

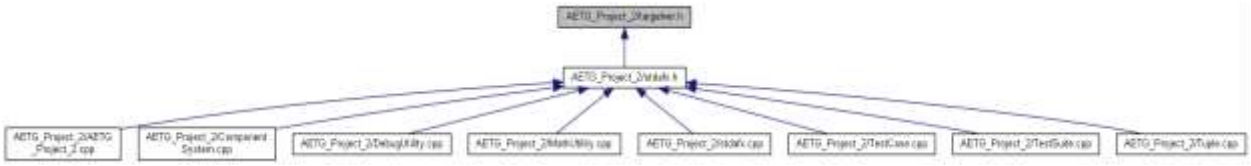
Windows OS platform header file.

```
#include <SDKDDKVer.h>
```

Include dependency graph for targetver.h:



This graph shows which files directly or indirectly include this file:



Detailed Description

Windows OS platform header file.

Including SDKDDKVer.h defines the highest available Windows platform. If you wish to build your application for a previous Windows platform, include WinSDKVer.h andset the _WIN32_WINNT macro to the platform you wish to support before including SDKDDKVer.h.

Author:

Mark L. Short

Date:

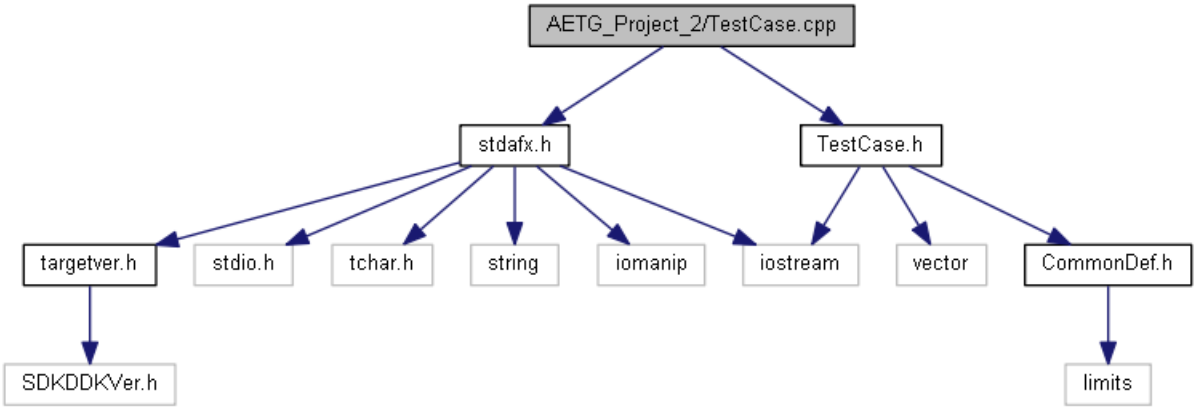
February 9, 2015

AETG_Project_2/TestCase.cpp File Reference

CTestCase class implementation.

```
#include "stdafx.h"
#include "TestCase.h"
```

Include dependency graph for TestCase.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const CTestCase &rhs)
  Overloaded stream extraction operator

Detailed Description

CTestCase class implementation.

Author:
Mark L. Short

Date:
February 9, 2015

Function Documentation

std::ostream& operator<< (std::ostream & *os*, const CTestCase & *rhs*)

Overloaded stream extraction operator

Parameters:

in,out	*os*	reference to an ostream object
in	*rhs*	target CTestCase object to be written to the stream

Return values:

tostream&	a reference to the resultant stream object

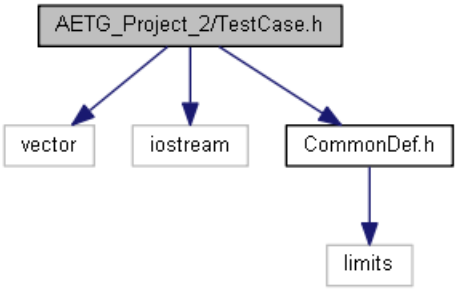
```
12 {
13     for (auto& it : rhs)
14         os << it <<  T(" ");
15
16     return os;
17 }
```

AETG_Project_2/TestCase.h File Reference

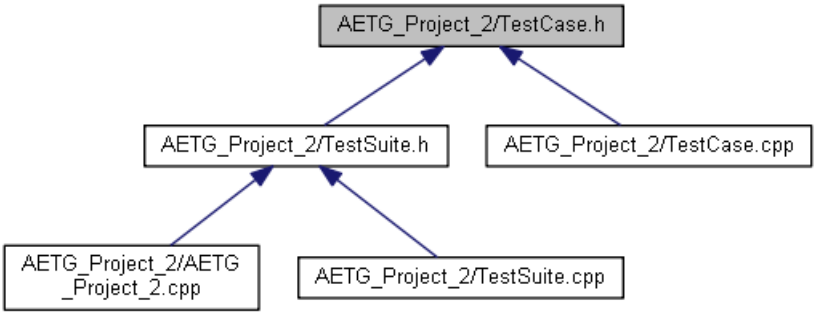
CTestCase class interface.

```
#include <vector>
#include <iostream>
#include "CommonDef.h"
```

Include dependency graph for TestCase.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CTestCase

A primitive test case class implementation. Macros

- #define **__TEST_CASE_H__**
- #define **tostream** std::ostream
- #define **tistream** std::istream

Functions

- std::ostream & operator<< (std::ostream &os, const CTestCase &rhs)
  overloaded stream extraction operator

Detailed Description

CTestCase class interface.

Author:
Mark L. Short

Date:
February 9, 2015

Function Documentation

std::ostream& operator<< (std::ostream & *os*, const CTestCase & *rhs*)

overloaded stream extraction operator

Parameters:

in,out	*os*	reference to an ostream object
in	*rhs*	target CTestCase object to be written to the stream

Return values:

tostream&	a reference to the resultant stream object

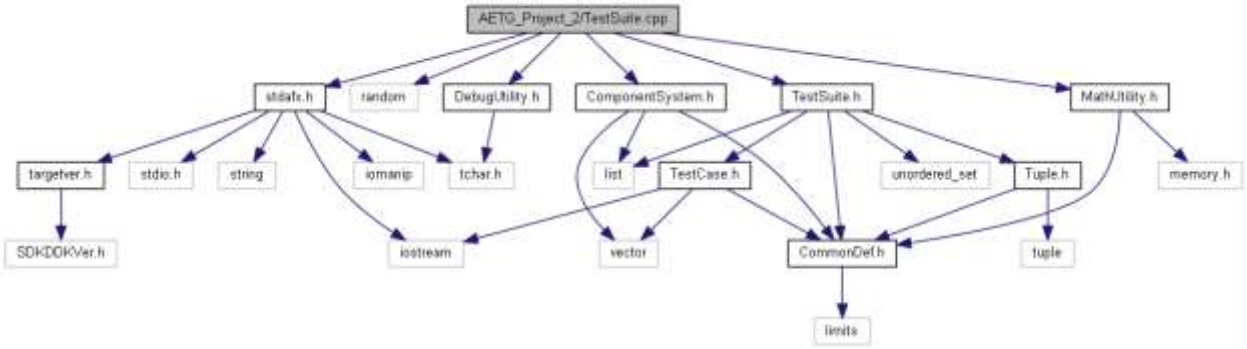
```
12 {
13     for (auto& it : rhs)
14         os << it <<  T(" ");
15
16     return os;
17 }
```

AETG_Project_2/TestSuite.cpp File Reference

CTestSuite class implementation.

```
#include "stdafx.h"
#include <random>
#include "ComponentSystem.h"
#include "DebugUtility.h"
#include "MathUtility.h"
#include "TestSuite.h"
```

Include dependency graph for TestSuite.cpp:



Functions

- LEVEL_T FindGreatestOccuringLevel (const std::vector< LEVEL_T > &vLevels)
  Returns the index of the largest level of a sequence.

Variables

- std::mt19937 g_mt
  the Mersenne Twister engine
- const int T_WAY = 3

Detailed Description

CTestSuite class implementation.

Author:
Mark L. Short

Date:
February 9, 2015

Function Documentation

LEVEL_T FindGreatestOccuringLevel (const std::vector< LEVEL_T > & *vLevels*)

Returns the index of the largest level of a sequence.

Parameters:

in	*vLevels*	target sequence of level values

Return values:

LEVEL_T	on success containing the index value
LEVEL_INVALID	on error

```
301 {
302     LEVEL_T nResult        = LEVEL_INVALID;
303
304     int     iMaxLevelCount = 0;
305     int     iNumLevels     = vLevels.size();
306     int     iRandomMod     = 2;
307
308     for (int i = 0; i < iNumLevels; i++)
309     {
310         if (vLevels[i] > iMaxLevelCount)
311         {
312             iMaxLevelCount = vLevels[i];
313             nResult        = i;
314         }
315         else if ((vLevels[i] == iMaxLevelCount) && (vLevels[i] > 0))
316         {   // some level of randomization to mix things up a bit
317             if ((rand() % iRandomMod) == 0)
318             {
319                 nResult = i;
320                 iRandomMod ++;
321             }
322         }
323     }
324
325     return nResult;
326 }
```

Variable Documentation

std::mt19937 g_mt

the Mersenne Twister engine

Todo:
refactor this

const int T_WAY = 3

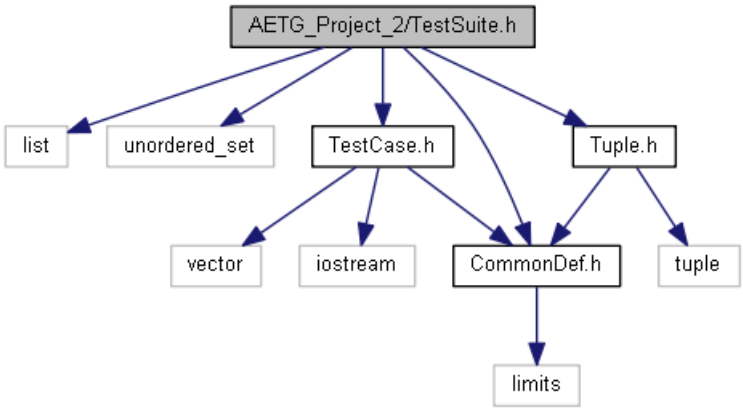
Todo:
make this configurable

AETG_Project_2/TestSuite.h File Reference

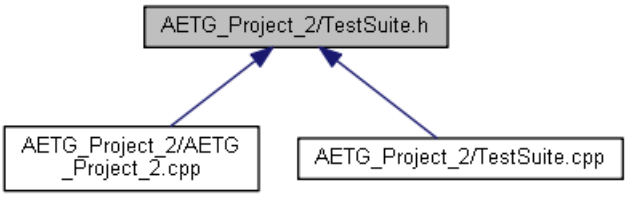
CTestSuite class interface.

```
#include <list>
#include <unordered_set>
#include "TestCase.h"
#include "Tuple.h"
#include "CommonDef.h"
```

Include dependency graph for TestSuite.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CTestSuite

Manages test case generation and the and the collection of test cases. Typedefs

- typedef std::unordered_set< T2_TUPLE, T2_TUPLE_HASH > T2_TUPLE_HASHSET
  A hash set type definition based on a T2_TUPLE hash algorithm.
- typedef std::unordered_set< T3_TUPLE, T3_TUPLE_HASH > T3_TUPLE_HASHSET
  A hash set type definition based on a T3_TUPLE hash algorithm.

Variables

- const int TEST_CASE_CANDIDATES = 50
  Global predetermined test case candidate generation.

Detailed Description

CTestSuite class interface.

Provides type definitions for T2_TUPLE_HASHSET, T3_TUPLE_HASHSET, CTestSuite

Author:

Mark L. Short

Date:

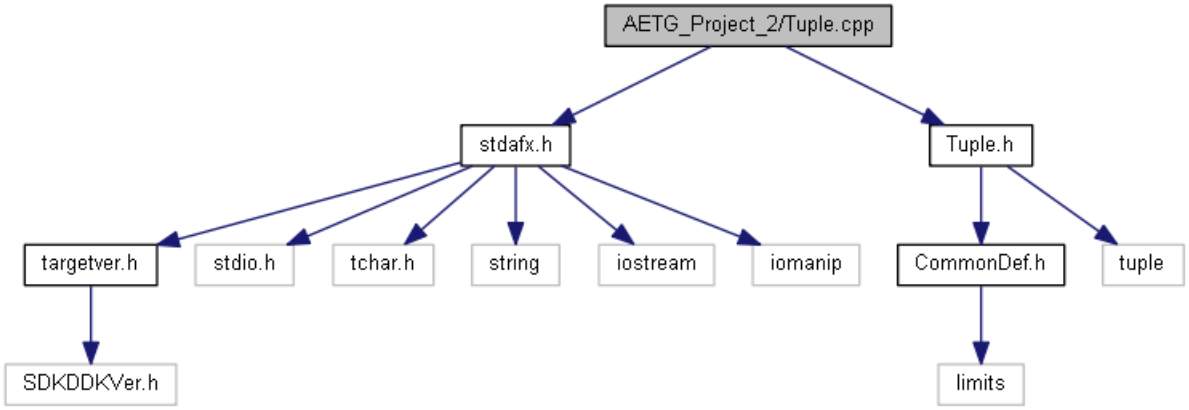
February 9, 2015

AETG_Project_2/Tuple.cpp File Reference

Tuple utility methods.

```
#include "stdafx.h"
#include "Tuple.h"
```

Include dependency graph for Tuple.cpp:



Functions

- void **PrintTuple** (const T2_TUPLE &tpl)
- void **PrintTuple** (const T3_TUPLE &tpl)

Detailed Description

Tuple utility methods.

Author:

Mark L. Short

Date:

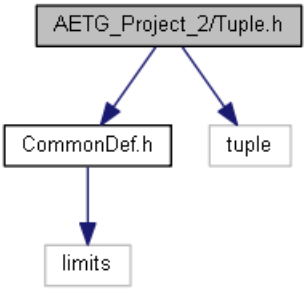
February 9, 2015

AETG_Project_2/Tuple.h File Reference

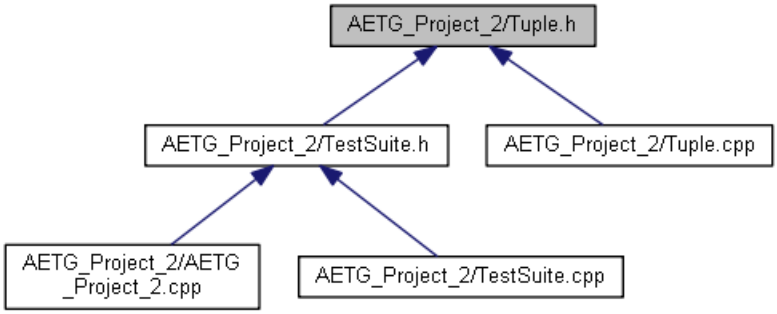
Various tuple type definitions & constructs.

```
#include "CommonDef.h"
#include <tuple>
```

Include dependency graph for Tuple.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct T2_TUPLE_HASH
- *A primitive hash function implementation.* struct T3_TUPLE_HASH

A primitive hash function implementation. Macros

- #define **__TUPLE_H__**
- #define LEVEL_BITS (sizeof(LEVEL_T) * CHAR_BIT)
  Calculate the number of bits contained in the LEVEL_T type

Typedefs

- typedef std::tuple< LEVEL_T, LEVEL_T > T2_TUPLE
  2-way tuple type
- typedef std::tuple< LEVEL_T, LEVEL_T, LEVEL_T > T3_TUPLE
  3-way tuple type

Functions

- void **PrintTuple** (const T2_TUPLE &tpl)
- void **PrintTuple** (const T3_TUPLE &tpl)

Detailed Description

Various tuple type definitions & constructs.

Provides type definitions for: T2_TUPLE, T3_TUPLE, T2_TUPLE_HASH, T3_TUPLE_HASH

Author:

Mark L. Short

Date:

February 9, 2015

Index