Samchon Framework for CPP

1.0.0

Generated by Doxygen 1.8.10

Sun Oct 25 2015 00:45:34

Contents

1 Samchon Framework **1**

2 Namespace Index **3**

2.1 Namespace List 3

3 Hierarchical Index **5**

3.1 Class Hierarchy 5

4 Class Index **9**

4.1 Class List 9

5 Namespace Documentation **15**

5.1 samchon Namespace Reference 15

5.1.1 Detailed Description 16

5.1.2 Typedef Documentation 16

5.1.2.1 Dictionary 16

5.2 samchon::example::chat_service Namespace Reference 17

5.2.1 Detailed Description 17

5.3 samchon::example::console_chat_server Namespace Reference 18

5.3.1 Detailed Description 18

5.4 samchon::example::tsp Namespace Reference 19

5.4.1 Detailed Description 19

5.5 samchon::library Namespace Reference 22

5.5.1 Detailed Description 24

5.5.2 Typedef Documentation 29

5.5.2.1 CriticalDictionary 29

5.5.2.2 CriticalList 29

5.5.2.3 CriticalMap 30

5.5.2.4 CriticalSet 30

5.5.2.5 CriticalVector 30

5.5.2.6 XMLList 30

5.6 samchon::namtree Namespace Reference 31

5.6.1 Detailed Description 31

5.7 samchon::protocol Namespace Reference . . . 35
5.7.1 Detailed Description . . . 37
5.7.2 Typedef Documentation . . . 46
5.7.2.1 SharedEntityArray . . . 46
5.7.2.2 SharedEntityList . . . 47
5.7.2.3 UniqueEntityArray . . . 47
5.7.2.4 UniqueEntityList . . . 47
5.8 samchon::protocol::master Namespace Reference . . . 48
5.8.1 Detailed Description . . . 49
5.9 samchon::protocol::service Namespace Reference . . . 51
5.9.1 Detailed Description . . . 52
5.10 samchon::protocol::slave Namespace Reference . . . 61
5.10.1 Detailed Description . . . 61

6 Class Documentation **63**
6.1 Base64 Class Reference . . . 63
6.1.1 Detailed Description . . . 63
6.1.2 Member Function Documentation . . . 64
6.1.2.1 encode(const ByteArray &) -> std::string . . . 64
6.1.2.2 decode(const std::string &) -> ByteArray . . . 64
6.2 ByteArray Class Reference . . . 65
6.2.1 Detailed Description . . . 66
6.2.2 Member Function Documentation . . . 66
6.2.2.1 getPosition() const -> size_t . . . 66
6.2.2.2 setPosition(size_t) . . . 66
6.2.2.3 reverse(const _Ty &val) -> _Ty . . . 67
6.2.2.4 read() const -> _Ty . . . 67
6.2.2.5 readReversely() const -> _Ty . . . 67
6.2.2.6 write(const _Ty &val) . . . 67
6.2.2.7 writeReversely(const _Ty &val) . . . 68
6.2.2.8 compress() const -> ByteArray . . . 68
6.2.2.9 decompress() const -> ByteArray . . . 68
6.2.3 Member Data Documentation . . . 68
6.2.3.1 position . . . 68
6.3 CaseGenerator Class Reference . . . 69
6.3.1 Detailed Description . . . 69
6.3.2 Constructor & Destructor Documentation . . . 78
6.3.2.1 CaseGenerator(size_t n, size_t r) . . . 78
6.3.3 Member Function Documentation . . . 78
6.3.3.1 size() const -> size_t . . . 78

6.3.3.2 operator[](size_t) const -> std::vector< size_t >=0 . . . 78
6.3.3.3 at(size_t) const -> std::vector< size_t > . . . 78
6.3.3.4 toMatrix() const -> std::vector< std::vector< size_t >> . . . 78
6.4 Charset Class Reference . . . 79
6.4.1 Detailed Description . . . 79
6.4.2 Member Function Documentation . . . 80
6.4.2.1 toMultibyte(const std::string &) -> std::string . . . 80
6.4.2.2 toMultibyte(const std::wstring &) -> std::string . . . 81
6.4.2.3 toUTF8(const std::string &) -> std::string . . . 81
6.4.2.4 toUTF8(const std::wstring &) -> std::string . . . 81
6.4.2.5 toUnicode(const std::string &, int) -> std::wstring . . . 82
6.5 ChatClient Class Reference . . . 82
6.5.1 Detailed Description . . . 83
6.5.2 Member Function Documentation . . . 83
6.5.2.1 createService(const std::string &) -> protocol::service::Service *override . . . 83
6.6 ChatClient Class Reference . . . 84
6.6.1 Detailed Description . . . 84
6.6.2 Constructor & Destructor Documentation . . . 85
6.6.2.1 ChatClient(ChatServer *, protocol::Socket *) . . . 85
6.6.2.2 ~ChatClient() . . . 85
6.6.3 Member Function Documentation . . . 85
6.6.3.1 replyData(std::shared_ptr< protocol::Invoke >) override . . . 85
6.7 ChatMessage Class Reference . . . 86
6.7.1 Detailed Description . . . 86
6.7.2 Member Function Documentation . . . 87
6.7.2.1 TAG() const -> std::string . . . 87
6.7.2.2 construct(std::shared_ptr< library::XML >) . . . 87
6.7.2.3 getListener() const -> std::string . . . 88
6.7.2.4 toXML() const -> std::shared_ptr< library::XML > . . . 88
6.7.3 Member Data Documentation . . . 88
6.7.3.1 listener . . . 88
6.8 ChatRoom Class Reference . . . 88
6.8.1 Detailed Description . . . 89
6.8.2 Constructor & Destructor Documentation . . . 90
6.8.2.1 ChatRoom(ChatRoomArray *, const std::string &, ChatUser *) . . . 90
6.8.3 Member Function Documentation . . . 90
6.8.3.1 TAG() const -> std::string . . . 90
6.8.3.2 registerClient(ChatService *) . . . 91
6.8.3.3 eraseClient(ChatService *) . . . 91
6.8.3.4 toXML() const -> std::shared_ptr< library::XML > override . . . 91

6.8.4 Member Data Documentation . . . 91
6.8.4.1 name . . . 91
6.9 ChatRoomArray Class Reference . . . 91
6.9.1 Detailed Description . . . 92
6.9.2 Member Function Documentation . . . 93
6.9.2.1 CHILD_TAG() const -> std::string . . . 93
6.9.2.2 notify() . . . 93
6.10 ChatServer Class Reference . . . 94
6.10.1 Detailed Description . . . 94
6.10.2 Member Function Documentation . . . 95
6.10.2.1 replyData(std::shared_ptr< protocol::Invoke >) override . . . 95
6.10.2.2 sendData(std::shared_ptr< protocol::Invoke >) override . . . 95
6.11 ChatServer Class Reference . . . 95
6.11.1 Detailed Description . . . 96
6.12 ChatService Class Reference . . . 97
6.12.1 Detailed Description . . . 97
6.12.2 Member Data Documentation . . . 98
6.12.2.1 room . . . 98
6.13 ChatUser Class Reference . . . 98
6.13.1 Detailed Description . . . 99
6.14 Client Class Reference . . . 100
6.14.1 Detailed Description . . . 102
6.14.2 Constructor & Destructor Documentation . . . 102
6.14.2.1 Client(User *) . . . 102
6.14.3 Member Function Documentation . . . 103
6.14.3.1 createService(const std::string &) -> Service *=0 . . . 103
6.14.3.2 sendData(std::shared_ptr< Invoke >) . . . 103
6.14.3.3 replyData(std::shared_ptr< Invoke >) . . . 103
6.14.3.4 constructService(const std::string &) . . . 105
6.15 CombinedPermutationGenerator Class Reference . . . 105
6.15.1 Detailed Description . . . 105
6.15.2 Constructor & Destructor Documentation . . . 106
6.15.2.1 CombinedPermutationGenerator(size_t n, size_t r) . . . 106
6.15.3 Member Function Documentation . . . 106
6.15.3.1 operator[](size_t) const -> std::vector< size_t > override . . . 106
6.16 CriticalAllocator< _Ty > Class Template Reference . . . 107
6.16.1 Detailed Description . . . 108
6.16.2 Member Function Documentation . . . 109
6.16.2.1 construct(_U *ptr, _Args &&...args) . . . 109
6.16.2.2 destroy(_U *ptr) . . . 110

6.16.2.3 allocate(size_type n, std::allocator< void >::const_pointer hint=NULL) -> pointer 110
6.16.2.4 deallocate(pointer ptr, size_type size) . . . 111
6.17 Date Class Reference . . . 111
6.17.1 Detailed Description . . . 113
6.17.2 Constructor & Destructor Documentation . . . 114
6.17.2.1 Date() . . . 114
6.17.2.2 Date(const super &) . . . 114
6.17.2.3 Date(super &&) . . . 114
6.17.2.4 Date(int year, int month, int date) . . . 114
6.17.2.5 Date(const WeakString &) . . . 114
6.17.2.6 Date(long long linuxTime) . . . 115
6.17.3 Member Function Documentation . . . 115
6.17.3.1 calcSeconds(int year, int month, int date) -> long long . . . 115
6.17.3.2 calcLastDates(int year) -> std::array< int, 12 > . . . 115
6.17.3.3 set(const std::string &) . . . 115
6.17.3.4 set(const WeakString &) . . . 115
6.17.3.5 set(int year, int month, int date) . . . 116
6.17.3.6 set(long long linuxTime) . . . 116
6.17.3.7 setYear(int) . . . 116
6.17.3.8 setMonth(int) . . . 116
6.17.3.9 setDate(int) . . . 117
6.17.3.10 addYear(int) . . . 117
6.17.3.11 addMonth(int) . . . 117
6.17.3.12 addWeek(int) . . . 117
6.17.3.13 addDate(int) . . . 117
6.17.3.14 getYear() const -> int . . . 118
6.17.3.15 getMonth() const -> int . . . 118
6.17.3.16 getDate() const -> int . . . 118
6.17.3.17 getDay() const -> int . . . 118
6.17.3.18 toTM() const -> struct::tm . . . 118
6.17.3.19 toLinuxTime() const -> long long . . . 118
6.17.3.20 toString() const -> std::string . . . 119
6.17.4 Member Data Documentation . . . 119
6.17.4.1 TP_1970 . . . 119
6.18 Datetime Class Reference . . . 119
6.18.1 Detailed Description . . . 120
6.18.2 Constructor & Destructor Documentation . . . 121
6.18.2.1 Datetime() . . . 121
6.18.2.2 Datetime(const super::super &) . . . 121
6.18.2.3 Datetime(super::super &&) . . . 122

6.18.2.4 Datetime(int year, int month, int date, int hour=0, int min=0, int sec=0) 122
6.18.2.5 Datetime(const WeakString &) 122
6.18.2.6 Datetime(long long linuxTime) 122
6.18.3 Member Function Documentation 122
6.18.3.1 set(const WeakString &) override 122
6.18.3.2 set(int year, int month, int date, int hours=0, int minutes=0, int seconds=0) 123
6.18.3.3 set(long long linuxTime) 123
6.18.3.4 setYear(int) 123
6.18.3.5 setMonth(int) 123
6.18.3.6 setDate(int) 123
6.18.3.7 setHour(int) 124
6.18.3.8 setMinute(int) 124
6.18.3.9 setSecond(int) 124
6.18.3.10 addYear(int) override 124
6.18.3.11 addMonth(int) override 124
6.18.3.12 addWeek(int) override 126
6.18.3.13 addDate(int) override 126
6.18.3.14 addHour(int) 126
6.18.3.15 addMinute(int) 126
6.18.3.16 addSecond(int) 127
6.18.3.17 getHour() const -> int 127
6.18.3.18 getMinute() const -> int 127
6.18.3.19 getSecond() const -> int 127
6.18.3.20 toString() const -> std::string override 127
6.19 DistributedClient Class Reference 128
6.19.1 Detailed Description 128
6.20 DistributedClientArray Class Reference 130
6.20.1 Detailed Description 131
6.20.2 Member Function Documentation 133
6.20.2.1 construct(std::shared_ptr< library::XML >) override 133
6.20.2.2 start() override 134
6.20.2.3 toXML() const -> std::shared_ptr< library::XML > override 134
6.21 DistributedClientArrayMediator Class Reference 135
6.21.1 Detailed Description 135
6.21.2 Member Function Documentation 138
6.21.2.1 construct(std::shared_ptr< library::XML >) override 138
6.21.2.2 start() override 139
6.21.2.3 toXML() const -> std::shared_ptr< library::XML > override 139
6.22 DistributedClientServerMediator Class Reference 140
6.22.1 Detailed Description 140

6.23 DistributedServer Class Reference . . . 143
6.23.1 Detailed Description . . . 143
6.23.2 Member Function Documentation . . . 145
6.23.2.1 construct(std::shared_ptr< library::XML >) override . . . 145
6.23.2.2 toXML() const -> std::shared_ptr< library::XML > override . . . 146
6.24 DistributedServerArray Class Reference . . . 146
6.24.1 Detailed Description . . . 147
6.24.2 Member Function Documentation . . . 149
6.24.2.1 start() override . . . 149
6.25 DistributedServerArrayMediator Class Reference . . . 149
6.25.1 Detailed Description . . . 150
6.25.2 Member Function Documentation . . . 153
6.25.2.1 start() override . . . 153
6.26 DistributedSlaveServerMediator Class Reference . . . 153
6.26.1 Detailed Description . . . 154
6.27 DistributedSlaveSystemMediator Class Reference . . . 157
6.27.1 Detailed Description . . . 157
6.27.2 Member Function Documentation . . . 160
6.27.2.1 replyData(std::shared_ptr< Invoke >) override . . . 160
6.27.2.2 TAG() const -> std::string override . . . 160
6.28 DistributedSystem Class Reference . . . 160
6.28.1 Detailed Description . . . 161
6.28.2 Member Function Documentation . . . 163
6.28.2.1 construct(std::shared_ptr< library::XML >) override . . . 163
6.28.2.2 sendData(std::shared_ptr< Invoke >) override . . . 164
6.28.2.3 replyData(std::shared_ptr< Invoke >) override . . . 164
6.28.2.4 toXML() const -> std::shared_ptr< library::XML > override . . . 164
6.28.3 Member Data Documentation . . . 165
6.28.3.1 inProgress . . . 165
6.28.3.2 avgElapsedTime . . . 165
6.28.3.3 performance . . . 165
6.29 DistributedSystemArray Class Reference . . . 165
6.29.1 Detailed Description . . . 166
6.29.2 Member Function Documentation . . . 168
6.29.2.1 construct(std::shared_ptr< library::XML >) override . . . 168
6.29.2.2 createRole(std::shared_ptr< library::XML >) -> DistributedSystemRole *=0 . . . 169
6.29.2.3 start() override . . . 169
6.29.2.4 allocateRoles() . . . 169
6.29.2.5 getRole(const std::string &) const -> std::shared_ptr< DistributedSystemRole > 169
6.29.2.6 toXML() const -> std::shared_ptr< library::XML > override . . . 170

6.30 DistributedSystemArrayMediator Class Reference . . . 170
6.30.1 Detailed Description . . . 171
6.30.2 Member Function Documentation . . . 174
6.30.2.1 construct(std::shared_ptr< library::XML >) override . . . 174
6.30.2.2 createSlave() const -> DistributedSlaveSystemMediator *=0 . . . 175
6.30.2.3 start() override . . . 175
6.30.2.4 toXML() const -> std::shared_ptr< library::XML > override . . . 175
6.31 DistributedSystemRole Class Reference . . . 176
6.31.1 Detailed Description . . . 177
6.31.2 Member Function Documentation . . . 178
6.31.2.1 getSystem() const -> ExternalSystem *=delete . . . 178
6.31.2.2 sendData(std::shared_ptr< Invoke >) override . . . 178
6.31.2.3 toXML() const -> std::shared_ptr< library::XML > override . . . 178
6.32 DSInvokeHistory Class Reference . . . 179
6.32.1 Detailed Description . . . 180
6.32.2 Constructor & Destructor Documentation . . . 181
6.32.2.1 DSInvokeHistory(DistributedSystem *, DistributedSystemRole *) . . . 181
6.33 DSInvokeHistoryList Class Reference . . . 181
6.33.1 Detailed Description . . . 182
6.33.2 Member Function Documentation . . . 182
6.33.2.1 createChild(std::shared_ptr< library::XML >) -> DSInvokeHistory *override . . . 182
6.34 Entity Class Reference . . . 183
6.34.1 Detailed Description . . . 183
6.34.2 Member Function Documentation . . . 187
6.34.2.1 TAG() const -> std::string=0 . . . 187
6.34.2.2 construct(std::shared_ptr< library::XML >)=0 . . . 188
6.34.2.3 key() const -> std::string . . . 188
6.34.2.4 toXML() const -> std::shared_ptr< library::XML > . . . 188
6.35 EntityArray< _Ty > Class Template Reference . . . 189
6.35.1 Detailed Description . . . 189
6.35.2 Member Function Documentation . . . 194
6.35.2.1 construct(std::shared_ptr< library::XML > xml) override . . . 194
6.35.2.2 toXML() const -> std::shared_ptr< library::XML > override . . . 194
6.36 EntityGroup< _Container, _ETy, _Ty > Class Template Reference . . . 195
6.36.1 Detailed Description . . . 196
6.36.2 Member Function Documentation . . . 203
6.36.2.1 construct(std::shared_ptr< library::XML > xml) . . . 203
6.36.2.2 createChild(std::shared_ptr< library::XML >) -> entity_type *=0 . . . 204
6.36.2.3 has(const std::string &key) const -> bool . . . 204
6.36.2.4 get(const std::string &key) -> value_type & . . . 204

6.36.2.5 get(const std::string &key) const -> const value_type & . . . 205
6.36.2.6 toXML() const -> std::shared_ptr< library::XML > . . . 206
6.37 EntityList< _Ty > Class Template Reference . . . 206
6.37.1 Detailed Description . . . 207
6.37.2 Member Function Documentation . . . 212
6.37.2.1 construct(std::shared_ptr< library::XML > xml) override . . . 212
6.37.2.2 toXML() const -> std::shared_ptr< library::XML > override . . . 212
6.38 ErrorEvent Class Reference . . . 213
6.38.1 Detailed Description . . . 214
6.38.2 Constructor & Destructor Documentation . . . 216
6.38.2.1 ErrorEvent(EventDispatcher *, const std::string &) . . . 216
6.39 Event Class Reference . . . 216
6.39.1 Detailed Description . . . 217
6.39.2 Constructor & Destructor Documentation . . . 219
6.39.2.1 Event(EventDispatcher *, int) . . . 219
6.40 EventDispatcher Class Reference . . . 219
6.40.1 Detailed Description . . . 220
6.40.2 Constructor & Destructor Documentation . . . 222
6.40.2.1 EventDispatcher(const EventDispatcher &eventDispatcher) . . . 222
6.40.2.2 EventDispatcher(EventDispatcher &&eventDispatcher) . . . 222
6.40.3 Member Function Documentation . . . 222
6.40.3.1 addEventListener(int, void(*listener)(std::shared_ptr< Event >)) . . . 222
6.40.3.2 removeEventListener(int, void(*listener)(std::shared_ptr< Event >)) . . . 222
6.40.3.3 dispatchEvent(std::shared_ptr< Event >) -> bool . . . 222
6.40.3.4 dispatchProgressEvent(size_t x, size_t size) -> bool . . . 223
6.41 ExternalClient Class Reference . . . 223
6.41.1 Detailed Description . . . 224
6.42 ExternalClientArray Class Reference . . . 225
6.42.1 Detailed Description . . . 227
6.42.2 Member Function Documentation . . . 228
6.42.2.1 construct(std::shared_ptr< library::XML >) override . . . 228
6.42.2.2 start() override . . . 228
6.42.2.3 toXML() const -> std::shared_ptr< library::XML > override . . . 228
6.43 ExternalServer Class Reference . . . 229
6.43.1 Detailed Description . . . 230
6.43.2 Member Function Documentation . . . 231
6.43.2.1 construct(std::shared_ptr< library::XML >) override . . . 231
6.43.2.2 start() override . . . 231
6.43.2.3 toXML() const -> std::shared_ptr< library::XML > override . . . 232
6.44 ExternalServerArray Class Reference . . . 232

6.44.1 Detailed Description . . . 233
6.44.2 Member Function Documentation . . . 234
6.44.2.1 start() override . . . 234
6.45 ExternalSystem Class Reference . . . 235
6.45.1 Detailed Description . . . 236
6.45.2 Member Function Documentation . . . 237
6.45.2.1 construct(std::shared_ptr< library::XML >) override . . . 237
6.45.2.2 start()=0 . . . 237
6.45.2.3 key() const -> std::string override . . . 237
6.45.2.4 replyData(std::shared_ptr< Invoke >) override . . . 237
6.45.2.5 TAG() const -> std::string override . . . 238
6.45.2.6 CHILD_TAG() const -> std::string override . . . 238
6.45.2.7 toXML() const -> std::shared_ptr< library::XML > override . . . 238
6.45.3 Member Data Documentation . . . 239
6.45.3.1 name . . . 239
6.46 ExternalSystemArray Class Reference . . . 239
6.46.1 Detailed Description . . . 240
6.46.2 Member Function Documentation . . . 240
6.46.2.1 start()=0 . . . 240
6.46.2.2 hasRole(const std::string &) const -> bool . . . 241
6.46.2.3 getRole(const std::string &) const -> std::shared_ptr< ExternalSystemRole > . . . 241
6.46.2.4 sendData(std::shared_ptr< Invoke >) override . . . 241
6.46.2.5 replyData(std::shared_ptr< Invoke >) override . . . 241
6.46.2.6 TAG() const -> std::string override . . . 242
6.46.2.7 CHILD_TAG() const -> std::string override . . . 242
6.47 ExternalSystemRole Class Reference . . . 242
6.47.1 Detailed Description . . . 244
6.47.2 Constructor & Destructor Documentation . . . 244
6.47.2.1 ExternalSystemRole(ExternalSystem *) . . . 244
6.47.3 Member Function Documentation . . . 245
6.47.3.1 construct(std::shared_ptr< library::XML >) override . . . 245
6.47.3.2 key() const -> std::string override . . . 245
6.47.3.3 sendData(std::shared_ptr< Invoke >) override . . . 245
6.47.3.4 TAG() const -> std::string override . . . 245
6.47.3.5 toXML() const -> std::shared_ptr< library::XML > override . . . 246
6.48 FactorialGenerator Class Reference . . . 247
6.48.1 Detailed Description . . . 247
6.48.2 Constructor & Destructor Documentation . . . 248
6.48.2.1 FactorialGenerator(size_t size) . . . 248
6.49 FlashPolicyServer Class Reference . . . 248

6.49.1 Detailed Description . . . 249
6.49.2 Constructor & Destructor Documentation . . . 249
6.49.2.1 FlashPolicyServer() . . . 249
6.49.2.2 FlashPolicyServer(std::shared_ptr< library::XML >) . . . 249
6.49.3 Member Function Documentation . . . 249
6.49.3.1 accept(Socket *) . . . 249
6.50 FTByteFile Class Reference . . . 250
6.50.1 Detailed Description . . . 250
6.50.2 Constructor & Destructor Documentation . . . 251
6.50.2.1 FTByteFile(FTFolder *) . . . 251
6.51 FTFactory Class Reference . . . 251
6.51.1 Detailed Description . . . 252
6.51.2 Member Data Documentation . . . 253
6.51.2.1 application . . . 253
6.51.2.2 category . . . 253
6.51.2.3 instanceMap . . . 253
6.52 FTFile Class Reference . . . 254
6.52.1 Detailed Description . . . 254
6.52.2 Constructor & Destructor Documentation . . . 255
6.52.2.1 FTFile(FTFolder *) . . . 255
6.52.3 Member Function Documentation . . . 255
6.52.3.1 toXML() const -> std::shared_ptr< library::XML > . . . 255
6.52.4 Member Data Documentation . . . 256
6.52.4.1 extension . . . 256
6.53 FTFolder Class Reference . . . 256
6.53.1 Detailed Description . . . 257
6.53.2 Constructor & Destructor Documentation . . . 258
6.53.2.1 FTFolder(FTFactory *, FTFolder *) . . . 258
6.53.3 Member Function Documentation . . . 258
6.53.3.1 CHILD_TAG() const -> std::string override . . . 258
6.53.3.2 toXML() const -> std::shared_ptr< library::XML > override . . . 259
6.54 FTInstance Class Reference . . . 259
6.54.1 Detailed Description . . . 260
6.54.2 Constructor & Destructor Documentation . . . 261
6.54.2.1 FTInstance(FTFolder *) . . . 261
6.54.3 Member Function Documentation . . . 261
6.54.3.1 TAG() const -> std::string . . . 261
6.54.3.2 key() const -> std::string . . . 262
6.54.3.3 toXML() const -> std::shared_ptr< library::XML > . . . 262
6.55 FTTextFile Class Reference . . . 262

6.55.1 Detailed Description . . . 263
6.55.2 Constructor & Destructor Documentation . . . 264
6.55.2.1 FTTextFile(FTFolder *) . . . 264
6.56 GAParameters Struct Reference . . . 264
6.56.1 Detailed Description . . . 265
6.56.2 Member Data Documentation . . . 265
6.56.2.1 mutationRate . . . 265
6.57 GAParameters Struct Reference . . . 265
6.57.1 Detailed Description . . . 266
6.58 GAPopulation< GeneArray, Compare > Class Template Reference . . . 266
6.58.1 Detailed Description . . . 267
6.58.2 Constructor & Destructor Documentation . . . 268
6.58.2.1 GAPopulation(size_t size) . . . 268
6.58.2.2 GAPopulation(std::shared_ptr< GeneArray > geneArray, size_t size) . . . 268
6.58.3 Member Function Documentation . . . 269
6.58.3.1 fitTest() const -> std::shared_ptr< GeneArray > . . . 269
6.59 GeneticAlgorithm< GeneArray, Compare > Class Template Reference . . . 269
6.59.1 Detailed Description . . . 270
6.59.2 Constructor & Destructor Documentation . . . 276
6.59.2.1 GeneticAlgorithm(bool unique, double mutationRate=0.015, size_t tournament=10) 276
6.59.3 Member Function Documentation . . . 276
6.59.3.1 evolveGeneArray(std::shared_ptr< GeneArray > geneArray, size_t population, size_t generation) const -> std::shared_ptr< GeneArray > . . . 276
6.59.3.2 evolvePopulation(std::shared_ptr< MyPopulation > population) const -> std↩::shared_ptr< MyPopulation > . . . 276
6.59.3.3 selection(std::shared_ptr< MyPopulation > population) const -> std::shared_↩ptr< GeneArray > . . . 276
6.59.3.4 crossover(std::shared_ptr< GeneArray > &parent1, std::shared_ptr< GeneArray > &parent2) const -> std::shared_ptr< GeneArray > . . . 277
6.59.3.5 mutate(std::shared_ptr< GeneArray > geneArray) const . . . 277
6.59.4 Member Data Documentation . . . 278
6.59.4.1 mutationRate . . . 278
6.60 GeometryPoint Class Reference . . . 278
6.60.1 Detailed Description . . . 279
6.60.2 Constructor & Destructor Documentation . . . 279
6.60.2.1 GeometryPoint(int) . . . 279
6.60.2.2 GeometryPoint(int, double, double) . . . 279
6.60.3 Member Function Documentation . . . 280
6.60.3.1 calcDistance(const GeometryPoint &) const -> double . . . 280
6.60.3.2 toString() const -> std::string . . . 281
6.61 HTTPLoader Class Reference . . . 281

6.61.1 Detailed Description . . . 281
6.62 IClient Class Reference . . . 282
6.62.1 Detailed Description . . . 283
6.62.2 Member Function Documentation . . . 284
6.62.2.1 listen() . . . 284
6.62.2.2 sendData(std::shared_ptr< Invoke >) . . . 284
6.62.2.3 _replyData(std::shared_ptr< Invoke >) . . . 285
6.63 IEntityChain Class Reference . . . 285
6.63.1 Detailed Description . . . 286
6.63.2 Constructor & Destructor Documentation . . . 286
6.63.2.1 IEntityChain(Entity *) . . . 286
6.64 IEntityGroup Class Reference . . . 287
6.64.1 Detailed Description . . . 287
6.64.2 Member Function Documentation . . . 287
6.64.2.1 CHILD_TAG() const -> std::string=0 . . . 287
6.65 IEntityPtrGroup< _Container > Class Template Reference . . . 288
6.65.1 Detailed Description . . . 288
6.66 IHTMLEntity Class Reference . . . 288
6.66.1 Detailed Description . . . 289
6.66.2 Member Function Documentation . . . 292
6.66.2.1 toTH(const _Ty &val, const _Args &...args) -> std::string . . . 292
6.66.2.2 toTR(const _Ty &val, const _Args &...args) -> std::string . . . 293
6.66.2.3 toHTML() const -> std::string=0 . . . 294
6.67 IndexPair< _Ty > Class Template Reference . . . 294
6.67.1 Detailed Description . . . 294
6.67.2 Member Function Documentation . . . 294
6.67.2.1 getIndex() const -> size_t . . . 294
6.67.2.2 getValue() -> _Ty & . . . 295
6.67.2.3 getValue() const -> const _Ty & . . . 295
6.68 Instance Class Reference . . . 295
6.68.1 Detailed Description . . . 296
6.68.2 Constructor & Destructor Documentation . . . 296
6.68.2.1 Instance(const std::string &name, int price, int volume, int weight) . . . 296
6.69 INTExplore Class Reference . . . 296
6.69.1 Detailed Description . . . 297
6.69.2 Member Function Documentation . . . 299
6.69.2.1 construct(std::shared_ptr< library::XML >) . . . 299
6.69.2.2 toXML() const -> std::shared_ptr< library::XML > . . . 299
6.70 Invoke Class Reference . . . 300
6.70.1 Detailed Description . . . 301

6.70.2 Constructor & Destructor Documentation . . . 303
6.70.2.1 Invoke(const std::string &listener) . . . 303
6.70.2.2 Invoke(const std::string &listener, const _Ty &val, const _Args &...args) . . . 303
6.70.3 Member Function Documentation . . . 304
6.70.3.1 construct(std::shared_ptr< library::XML >) override . . . 304
6.70.3.2 createChild(std::shared_ptr< library::XML >) -> InvokeParameter *override . . . 304
6.70.3.3 TAG() const -> std::string override . . . 305
6.70.3.4 CHILD_TAG() const -> std::string override . . . 305
6.70.3.5 toXML() const -> std::shared_ptr< library::XML > override . . . 305
6.71 InvokeHistory Class Reference . . . 306
6.71.1 Detailed Description . . . 307
6.71.2 Constructor & Destructor Documentation . . . 308
6.71.2.1 InvokeHistory(std::shared_ptr< Invoke >) . . . 308
6.71.3 Member Function Documentation . . . 308
6.71.3.1 construct(std::shared_ptr< library::XML >) override . . . 308
6.71.3.2 notifyEnd() . . . 308
6.71.3.3 key() const -> std::string override . . . 308
6.71.3.4 calcElapsedTime() const -> long long . . . 308
6.71.3.5 TAG() const -> std::string override . . . 309
6.71.3.6 toXML() const -> std::shared_ptr< library::XML > override . . . 309
6.71.3.7 toInvoke() const -> std::shared_ptr< Invoke > . . . 309
6.71.4 Member Data Documentation . . . 309
6.71.4.1 listener . . . 309
6.71.4.2 startTime . . . 310
6.71.4.3 endTime . . . 310
6.72 InvokeHistoryArray Class Reference . . . 310
6.72.1 Detailed Description . . . 311
6.72.2 Member Function Documentation . . . 311
6.72.2.1 createChild(std::shared_ptr< library::XML >) -> InvokeHistory *override . . . 311
6.72.2.2 TAG() const -> std::string . . . 311
6.72.2.3 CHILD_TAG() const -> std::string . . . 311
6.73 InvokeParameter Class Reference . . . 312
6.73.1 Detailed Description . . . 313
6.73.2 Constructor & Destructor Documentation . . . 314
6.73.2.1 InvokeParameter(const std::string &, const std::string &, const std::string &) . . . 314
6.73.2.2 InvokeParameter(const std::string &name, const _Ty &val) . . . 315
6.73.3 Member Function Documentation . . . 315
6.73.3.1 construct(std::shared_ptr< library::XML >) override . . . 315
6.73.3.2 key() const -> std::string override . . . 315
6.73.3.3 getValue() const -> _Ty . . . 316

6.73.3.4 getvalueAsXML() const -> std::shared_ptr< library::XML > . . . 316
6.73.3.5 referValue() const -> const _Ty & . . . 316
6.73.3.6 moveValue() -> _Ty . . . 316
6.73.3.7 TAG() const -> std::string override . . . 316
6.73.3.8 toXML() const -> std::shared_ptr< library::XML > override . . . 316
6.73.4 Member Data Documentation . . . 317
6.73.4.1 type . . . 317
6.74 IOperator< _Ty > Class Template Reference . . . 317
6.74.1 Detailed Description . . . 317
6.74.2 Member Function Documentation . . . 318
6.74.2.1 operator<(const IOperator &) const -> bool=0 . . . 318
6.74.2.2 operator==(const IOperator &) const -> bool=0 . . . 318
6.75 IProtocol Class Reference . . . 318
6.75.1 Detailed Description . . . 319
6.75.2 Member Function Documentation . . . 319
6.75.2.1 replyData(std::shared_ptr< Invoke >) . . . 319
6.75.2.2 sendData(std::shared_ptr< Invoke >) . . . 320
6.76 IPUserPair Class Reference . . . 320
6.76.1 Detailed Description . . . 321
6.76.2 Member Function Documentation . . . 322
6.76.2.1 getSessionID(Socket *, size_t) -> std::string . . . 322
6.76.2.2 issueSessionID(size_t) const -> std::string . . . 322
6.77 IServer Class Reference . . . 322
6.77.1 Detailed Description . . . 323
6.77.2 Member Function Documentation . . . 326
6.77.2.1 open() . . . 326
6.78 ISQLEntity Class Reference . . . 327
6.78.1 Detailed Description . . . 327
6.78.2 Member Function Documentation . . . 328
6.78.2.1 load(std::shared_ptr< library::SQLStatement > stmt) . . . 328
6.78.2.2 archive(std::shared_ptr< library::SQLStatement > stmt) . . . 328
6.78.2.3 toSQL() const -> std::string . . . 328
6.79 IWebClient Class Reference . . . 328
6.79.1 Detailed Description . . . 329
6.79.2 Member Function Documentation . . . 330
6.79.2.1 listen() override . . . 330
6.79.2.2 sendData(std::shared_ptr< Invoke >) override . . . 330
6.80 IWebServer Class Reference . . . 331
6.80.1 Detailed Description . . . 331
6.80.2 Member Function Documentation . . . 334

6.80.2.1 open() override . . . 334
6.81 ListService Class Reference . . . 335
6.81.1 Detailed Description . . . 335
6.81.2 Member Function Documentation . . . 336
6.81.2.1 createRoom(const std::string &) . . . 336
6.82 Map< _Kty, _Ty, _Pr, _Alloc > Class Template Reference . . . 336
6.82.1 Detailed Description . . . 337
6.82.2 Member Function Documentation . . . 338
6.82.2.1 has(const _Kty &key) const -> bool . . . 338
6.82.2.2 get(const _Kty &key) -> _Ty & . . . 338
6.82.2.3 set(const _Kty &key, const _Ty &val) . . . 339
6.82.2.4 pop(const _Kty &key) -> _Ty . . . 339
6.83 Math Class Reference . . . 339
6.83.1 Detailed Description . . . 340
6.83.2 Member Function Documentation . . . 341
6.83.2.1 random() -> double . . . 341
6.83.2.2 minimum(const _Cont &container) -> IndexPair< _Ty > . . . 341
6.83.2.3 maximum(const _Cont &container) -> IndexPair< _Ty > . . . 342
6.83.2.4 mean(const _Cont &container) -> double . . . 342
6.83.2.5 median(const _Cont &container) -> double . . . 342
6.83.2.6 mode(const _Cont &container) -> _Ty . . . 343
6.83.2.7 stdev_p(const _Cont &container) -> double . . . 343
6.83.2.8 stdev_s(const _Cont &container) -> double . . . 343
6.83.2.9 variance_p(const _Cont &container) -> double . . . 344
6.83.2.10 variance_s(const _Cont &container) -> double . . . 344
6.83.3 Member Data Documentation . . . 344
6.83.3.1 E . . . 344
6.83.3.2 PI . . . 345
6.83.3.3 LN2 . . . 345
6.83.3.4 LN10 . . . 345
6.83.3.5 LOG2E . . . 345
6.83.3.6 LOG10E . . . 345
6.83.3.7 SQRT1_2 . . . 345
6.83.3.8 SQRT2 . . . 345
6.84 MySQLi Class Reference . . . 346
6.84.1 Detailed Description . . . 346
6.84.2 Constructor & Destructor Documentation . . . 347
6.84.2.1 MySQLi(int port=3306) . . . 347
6.85 NTCriteria Class Reference . . . 348
6.85.1 Detailed Description . . . 348

6.85.2 Member Function Documentation . . . 351
6.85.2.1 construct(std::shared_ptr< library::XML >) override . . . 351
6.85.2.2 createChild(std::shared_ptr< library::XML >) -> NTCriteria *override . . . 351
6.85.2.3 TAG() const -> std::string override . . . 351
6.85.2.4 CHILD_TAG() const -> std::string override . . . 351
6.85.2.5 toXML() const -> std::shared_ptr< library::XML > override . . . 352
6.86 NTEntityGroup Class Reference . . . 352
6.86.1 Detailed Description . . . 352
6.86.2 Member Function Documentation . . . 353
6.86.2.1 begin() const -> std::shared_ptr< NTIterator >=0 . . . 353
6.86.2.2 end() const -> std::shared_ptr< NTIterator >=0 . . . 353
6.87 NTFactory Class Reference . . . 354
6.87.1 Detailed Description . . . 355
6.87.2 Constructor & Destructor Documentation . . . 356
6.87.2.1 NTFactory(NTEntityGroup *) . . . 356
6.87.3 Member Function Documentation . . . 356
6.87.3.1 createCriteria(NTCriteria *, std::shared_ptr< library::XML >) -> NTCriteria * . . . 356
6.87.3.2 createSide(std::shared_ptr< library::XML >) -> NTSide * . . . 356
6.88 NTFile Class Reference . . . 357
6.88.1 Detailed Description . . . 357
6.88.2 Constructor & Destructor Documentation . . . 359
6.88.2.1 NTFile(NTFactory *, library::FTFolder *) . . . 359
6.88.3 Member Function Documentation . . . 360
6.88.3.1 construct(std::shared_ptr< library::XML >) override . . . 360
6.88.3.2 toXML() const -> std::shared_ptr< library::XML > override . . . 360
6.88.4 Member Data Documentation . . . 360
6.88.4.1 factory . . . 360
6.88.4.2 otherside . . . 360
6.89 NTIterator Class Reference . . . 361
6.89.1 Detailed Description . . . 361
6.89.2 Member Function Documentation . . . 362
6.89.2.1 operator--() const -> std::shared_ptr< NTIterator >=0 . . . 362
6.89.2.2 operator++() const -> std::shared_ptr< NTIterator >=0 . . . 362
6.89.2.3 operator==(const NTIterator &) const -> bool=0 . . . 362
6.89.2.4 operator<(const NTIterator &) const -> bool=0 . . . 363
6.90 NTParameter Class Reference . . . 363
6.90.1 Detailed Description . . . 364
6.90.2 Member Function Documentation . . . 365
6.90.2.1 construct(std::shared_ptr< library::XML >) override . . . 365

6.90.2.2 createChild(std::shared_ptr< library::XML >) -> NTParameterDetermined *override . . . 365
6.90.2.3 key() const -> std::string override . . . 365
6.90.2.4 TAG() const -> std::string override . . . 365
6.90.2.5 CHILD_TAG() const -> std::string override . . . 366
6.90.2.6 toXML() const -> std::shared_ptr< library::XML > override . . . 366
6.90.3 Member Data Documentation . . . 366
6.90.3.1 name . . . 366
6.91 NTParameterArray Class Reference . . . 366
6.91.1 Detailed Description . . . 367
6.91.2 Member Function Documentation . . . 368
6.91.2.1 TAG() const -> std::string override . . . 368
6.91.2.2 CHILD_TAG() const -> std::string override . . . 368
6.91.2.3 createChild(std::shared_ptr< library::XML >) -> NTParameter *override . . . 369
6.92 NTParameterDetermined Class Reference . . . 369
6.92.1 Detailed Description . . . 370
6.92.2 Member Function Documentation . . . 371
6.92.2.1 construct(std::shared_ptr< library::XML >) override . . . 371
6.92.2.2 key() const -> std::string override . . . 371
6.92.2.3 TAG() const -> std::string override . . . 371
6.92.2.4 toXML() const -> std::shared_ptr< library::XML > override . . . 371
6.92.3 Member Data Documentation . . . 372
6.92.3.1 label . . . 372
6.93 NTSide Class Reference . . . 372
6.93.1 Detailed Description . . . 373
6.93.2 Member Function Documentation . . . 374
6.93.2.1 TAG() const -> std::string override . . . 374
6.93.2.2 construct(std::shared_ptr< library::XML >) override . . . 374
6.93.2.3 toXML() const -> std::shared_ptr< library::XML > override . . . 375
6.94 Packer Class Reference . . . 375
6.94.1 Detailed Description . . . 376
6.94.2 Constructor & Destructor Documentation . . . 376
6.94.2.1 Packer(std::vector< Product > *, std::vector< Wrapper > *) . . . 376
6.94.2.2 Packer(const Packer &) . . . 377
6.95 ParallelClient Class Reference . . . 377
6.95.1 Detailed Description . . . 378
6.96 ParallelClient Class Reference . . . 379
6.96.1 Detailed Description . . . 379
6.97 ParallelClientArray Class Reference . . . 380
6.97.1 Detailed Description . . . 381

6.98 ParallelClientArrayMediator Class Reference . . . 383
6.98.1 Detailed Description . . . 384
6.98.2 Member Function Documentation . . . 387
6.98.2.1 construct(std::shared_ptr< library::XML >) override . . . 387
6.98.2.2 start() override . . . 388
6.98.2.3 toXML() const -> std::shared_ptr< library::XML > override . . . 388
6.99 ParallelServer Class Reference . . . 389
6.99.1 Detailed Description . . . 389
6.100 ParallelServer Class Reference . . . 391
6.100.1 Detailed Description . . . 391
6.100.2 Member Function Documentation . . . 392
6.100.2.1 construct(std::shared_ptr< library::XML >) override . . . 392
6.100.2.2 toXML() const -> std::shared_ptr< library::XML > override . . . 393
6.101 ParallelServerArray Class Reference . . . 393
6.101.1 Detailed Description . . . 394
6.102 ParallelServerArrayMediator Class Reference . . . 396
6.102.1 Detailed Description . . . 397
6.102.2 Member Function Documentation . . . 400
6.102.2.1 start() override . . . 400
6.103 ParallelSlaveClientMediator Class Reference . . . 400
6.103.1 Detailed Description . . . 401
6.104 ParallelSlaveServerMediator Class Reference . . . 404
6.104.1 Detailed Description . . . 404
6.105 ParallelSlaveSystemMediator Class Reference . . . 407
6.105.1 Detailed Description . . . 407
6.105.2 Member Function Documentation . . . 410
6.105.2.1 replyData(std::shared_ptr< Invoke >) override . . . 410
6.105.2.2 TAG() const -> std::string override . . . 410
6.106 ParallelSystem Class Reference . . . 410
6.106.1 Detailed Description . . . 411
6.106.2 Member Function Documentation . . . 412
6.106.2.1 construct(std::shared_ptr< library::XML >) override . . . 412
6.106.2.2 _replyData(std::shared_ptr< Invoke >) override . . . 413
6.106.2.3 sendSegmentData(std::shared_ptr< Invoke >, size_t, size_t) . . . 413
6.106.2.4 toXML() const -> std::shared_ptr< library::XML > override . . . 413
6.106.3 Member Data Documentation . . . 414
6.106.3.1 performance . . . 414
6.107 ParallelSystem Class Reference . . . 414
6.107.1 Detailed Description . . . 415
6.107.2 Member Function Documentation . . . 416

6.107.2.1 _replyData(std::shared_ptr< Invoke >) override . . . 416
6.107.2.2 replyData(std::shared_ptr< Invoke >, size_t, size_t)=0 . . . 417
6.108 ParallelSystemArray Class Reference . . . 417
6.108.1 Detailed Description . . . 418
6.108.2 Member Function Documentation . . . 419
6.108.2.1 sendSegmentData(std::shared_ptr< Invoke >, size_t) . . . 419
6.109 ParallelSystemArrayMediator Class Reference . . . 419
6.109.1 Detailed Description . . . 420
6.109.2 Member Function Documentation . . . 423
6.109.2.1 construct(std::shared_ptr< library::XML >) override . . . 423
6.109.2.2 createSlave() const -> ParallelSlaveSystemMediator *=0 . . . 423
6.109.2.3 start() override . . . 424
6.109.2.4 toXML() const -> std::shared_ptr< library::XML > override . . . 424
6.110 PermutationGenerator Class Reference . . . 424
6.110.1 Detailed Description . . . 425
6.110.2 Constructor & Destructor Documentation . . . 426
6.110.2.1 PermutationGenerator(size_t n, size_t r) . . . 426
6.110.3 Member Function Documentation . . . 426
6.110.3.1 operator[](size_t) const -> std::vector< size_t > override . . . 426
6.111 PRInvokeHistory Class Reference . . . 427
6.111.1 Detailed Description . . . 427
6.111.2 Constructor & Destructor Documentation . . . 428
6.111.2.1 PRInvokeHistory(std::shared_ptr< Invoke >) . . . 428
6.111.3 Member Function Documentation . . . 429
6.111.3.1 construct(std::shared_ptr< library::XML >) override . . . 429
6.111.3.2 toXML() const -> std::shared_ptr< library::XML > override . . . 429
6.112 PRInvokeHistoryArray Class Reference . . . 429
6.112.1 Detailed Description . . . 430
6.112.2 Member Function Documentation . . . 430
6.112.2.1 createChild(std::shared_ptr< library::XML >) -> InvokeHistory *override . . . 430
6.113 Product Class Reference . . . 431
6.113.1 Detailed Description . . . 431
6.113.2 Constructor & Destructor Documentation . . . 432
6.113.2.1 Product(const std::string &name, int price, int volume, int weight) . . . 432
6.114 ProgressEvent Class Reference . . . 432
6.114.1 Detailed Description . . . 433
6.114.2 Constructor & Destructor Documentation . . . 433
6.114.2.1 ProgressEvent(EventDispatcher *, size_t, size_t) . . . 433
6.114.3 Member Data Documentation . . . 434
6.114.3.1 denominator . . . 434

6.115 RWMutex Class Reference . . . 434
6.115.1 Detailed Description . . . 434
6.115.2 Member Function Documentation . . . 435
6.115.2.1 readLock() const . . . 435
6.115.2.2 readUnlock() const . . . 435
6.115.2.3 writeLock() . . . 436
6.116 Scheduler Class Reference . . . 436
6.116.1 Detailed Description . . . 437
6.116.2 Member Function Documentation . . . 437
6.116.2.1 optimize() -> std::shared_ptr< Travel > . . . 437
6.117 Semaphore Class Reference . . . 437
6.117.1 Detailed Description . . . 437
6.117.2 Constructor & Destructor Documentation . . . 438
6.117.2.1 Semaphore(size_t=2) . . . 438
6.117.3 Member Function Documentation . . . 439
6.117.3.1 setSize(size_t) . . . 439
6.117.3.2 size() const -> size_t . . . 439
6.117.3.3 acquire() . . . 439
6.117.3.4 tryAcquire() -> bool . . . 439
6.117.3.5 release() . . . 439
6.117.4 Member Data Documentation . . . 439
6.117.4.1 size_ . . . 439
6.117.4.2 mtx . . . 439
6.118 Server Class Reference . . . 440
6.118.1 Detailed Description . . . 441
6.118.2 Member Function Documentation . . . 441
6.118.2.1 begin() const -> const_iterator . . . 441
6.118.2.2 end() const -> const_iterator . . . 442
6.119 ServerConnector Class Reference . . . 442
6.119.1 Detailed Description . . . 443
6.119.2 Member Function Documentation . . . 443
6.119.2.1 connect() . . . 443
6.119.3 Member Data Documentation . . . 444
6.119.3.1 localEndPoint . . . 444
6.120 Service Class Reference . . . 444
6.120.1 Detailed Description . . . 445
6.120.2 Constructor & Destructor Documentation . . . 445
6.120.2.1 Service(Client *) . . . 445
6.120.3 Member Function Documentation . . . 445
6.120.3.1 sendData(std::shared_ptr< Invoke >) . . . 445

6.121 ServiceKeeper Class Reference . . . 446
6.121.1 Detailed Description . . . 446
6.121.2 Constructor & Destructor Documentation . . . 447
6.121.2.1 ServiceKeeper(const User ∗user, const Client ∗client) . . . 447
6.122 SharedAcquire Class Reference . . . 447
6.122.1 Detailed Description . . . 448
6.122.2 Constructor & Destructor Documentation . . . 449
6.122.2.1 SharedAcquire(Semaphore &, bool=true) . . . 449
6.122.3 Member Function Documentation . . . 449
6.122.3.1 acquire() . . . 449
6.122.3.2 release() . . . 450
6.123 SharedReadLock Class Reference . . . 450
6.123.1 Detailed Description . . . 450
6.123.2 Constructor & Destructor Documentation . . . 452
6.123.2.1 SharedReadLock(const RWMutex &, bool=true) . . . 452
6.123.3 Member Function Documentation . . . 453
6.123.3.1 lock() const . . . 453
6.123.3.2 unlock() const . . . 453
6.124 SharedWriteLock Class Reference . . . 453
6.124.1 Detailed Description . . . 454
6.124.2 Constructor & Destructor Documentation . . . 455
6.124.2.1 SharedWriteLock(RWMutex &, bool=true) . . . 455
6.124.3 Member Function Documentation . . . 455
6.124.3.1 lock() . . . 455
6.124.3.2 unlock() . . . 456
6.125 SlaveClient Class Reference . . . 456
6.125.1 Detailed Description . . . 457
6.126 SlaveServer Class Reference . . . 458
6.126.1 Detailed Description . . . 459
6.127 SlaveSystem Class Reference . . . 460
6.127.1 Detailed Description . . . 461
6.127.2 Member Function Documentation . . . 462
6.127.2.1 _replyData(std::shared_ptr< Invoke >) override . . . 462
6.128 SmartPointer< _Ty > Class Template Reference . . . 462
6.128.1 Detailed Description . . . 463
6.128.2 Constructor & Destructor Documentation . . . 463
6.128.2.1 SmartPointer() . . . 463
6.128.2.2 SmartPointer(const _Ty ∗ptr) . . . 463
6.128.2.3 SmartPointer(const SmartPointer &smartPointer) . . . 463
6.128.2.4 SmartPointer(SmartPointer &&smartPointer) . . . 463

6.128.2.5 ~SmartPointer() . . . 464
6.128.3 Member Function Documentation . . . 464
6.128.3.1 reset(const _Ty *ptr) . . . 464
6.128.3.2 get() const -> _Ty * . . . 464
6.128.3.3 operator->() const -> _Ty * . . . 464
6.128.3.4 operator*() const -> _Ty & . . . 464
6.129 SQLi Class Reference . . . 465
6.129.1 Detailed Description . . . 465
6.129.2 Constructor & Destructor Documentation . . . 466
6.129.2.1 SQLi(const std::string &driver, int port) . . . 466
6.129.3 Member Function Documentation . . . 467
6.129.3.1 connect(const std::string &ip, const std::string &db, const std::string &id, const std::string &pwd) . . . 467
6.129.3.2 createStatement() -> std::shared_ptr< SQLStatement > . . . 467
6.129.4 Member Data Documentation . . . 467
6.129.4.1 driver . . . 467
6.129.4.2 port . . . 467
6.130 SQLStatement Class Reference . . . 467
6.130.1 Detailed Description . . . 469
6.130.2 Constructor & Destructor Documentation . . . 470
6.130.2.1 SQLStatement(SQLi *sqli) . . . 470
6.130.3 Member Function Documentation . . . 470
6.130.3.1 prepare(const std::string &sql, const _Ty &val, const _Args &...args) . . . 470
6.130.3.2 execute() . . . 470
6.130.3.3 executeDirectly(const std::string &) . . . 470
6.130.3.4 fetch() const -> bool . . . 471
6.130.3.5 next() const -> bool . . . 471
6.130.3.6 size() const -> size_t . . . 471
6.130.3.7 at(size_t index) const -> _Ty . . . 471
6.130.3.8 get(const std::string &) const -> _Ty . . . 471
6.130.3.9 toXML() const -> std::shared_ptr< XML > . . . 472
6.130.4 Member Data Documentation . . . 472
6.130.4.1 bindParameterBASizeMap . . . 472
6.131 StringUtil Class Reference . . . 472
6.131.1 Detailed Description . . . 473
6.131.2 Member Function Documentation . . . 474
6.131.2.1 substitute(const std::string &format, const _Ty &val, const _Args &...args) -> std::string . . . 474
6.131.2.2 substituteSQL(const std::string &format, const _Ty &value, const _Args &...args) -> std::string . . . 475
6.131.2.3 isNumeric(const std::string &str) -> bool . . . 475

6.131.2.4 toNumber(const std::string &str) -> double . . . 475
6.131.2.5 numberFormat(double val, int precision=2) -> std::string . . . 475
6.131.2.6 percentFormat(double val, int precision=2) -> std::string . . . 476
6.131.2.7 colorNumberFormat(double value, int precision=2, double delimiter=0.0) -> std↩ ::string . . . 476
6.131.2.8 colorPercentFormat(double value, int precision=2, double delimiter=0.0) -> std↩ ::string . . . 476
6.131.2.9 trim(const std::string &val, const std::vector< std::string > &delims) -> std::string 476
6.131.2.10 ltrim(const std::string &val, const std::vector< std::string > &delims) -> std::string 477
6.131.2.11 rtrim(const std::string &val, const std::vector< std::string > &delims) -> std::string 477
6.131.2.12 finds(const std::string &str, const std::vector< std::string > &delims, size_t start↩ Index=0) -> IndexPair< std::string > . . . 477
6.131.2.13 rfinds(const std::string &str, const std::vector< std::string > &delims, size_t end↩ Index=SIZE_MAX) -> IndexPair< std::string > . . . 477
6.131.2.14 substring(const std::string &str, size_t startIndex, size_t endIndex=SIZE_MAX) - > std::string . . . 478
6.131.2.15 between(const std::string &str, const std::string &start="""", const std::string &end="""") -> std::string . . . 478
6.131.2.16 addTab(const std::string &str, size_t n=1) -> std::string . . . 478
6.131.2.17 split(const std::string &str, const std::string &delim) -> std::vector< std::string > 479
6.131.2.18 betweens(const std::string &str, const std::string &start="""", const std::string &end="""") -> std::vector< std::string > . . . 479
6.131.2.19 toLowerCase(const std::string &str) -> std::string . . . 479
6.131.2.20 toUpperCase(const std::string &str) -> std::string . . . 479
6.131.2.21 replaceAll(const std::string &str, const std::string &before, const std::string &after) -> std::string . . . 480
6.131.2.22 replaceAll(const std::string &str, const std::vector< std::pair< std::string, std↩ ::string >> &pairs) -> std::string . . . 480
6.132 SystemRole Class Reference . . . 480
6.132.1 Detailed Description . . . 481
6.132.2 Member Function Documentation . . . 481
6.132.2.1 construct(std::shared_ptr< library::XML >) override . . . 481
6.132.2.2 key() const -> std::string override . . . 481
6.132.2.3 TAG() const -> std::string override . . . 481
6.132.2.4 toXML() const -> std::shared_ptr< library::XML > override . . . 482
6.133 Travel Class Reference . . . 482
6.133.1 Detailed Description . . . 482
6.133.2 Constructor & Destructor Documentation . . . 483
6.133.2.1 Travel(const Travel &) . . . 483
6.133.3 Member Function Documentation . . . 483
6.133.3.1 operator<(const Travel &) const -> bool . . . 483
6.133.3.2 toString() const -> std::string . . . 483
6.133.4 Member Data Documentation . . . 484

6.133.4.1 distance . . . 484
6.134 TSQLi Class Reference . . . 484
6.134.1 Detailed Description . . . 485
6.134.2 Constructor & Destructor Documentation . . . 486
6.134.2.1 TSQLi(int port=1433) . . . 486
6.134.3 Member Function Documentation . . . 486
6.134.3.1 createStatement() -> std::shared_ptr< SQLStatement > . . . 486
6.135 TSQLStatement Class Reference . . . 486
6.135.1 Detailed Description . . . 487
6.135.2 Member Function Documentation . . . 488
6.135.2.1 toXML() const -> std::shared_ptr< XML > override . . . 488
6.136 UniqueAcquire Class Reference . . . 488
6.136.1 Detailed Description . . . 489
6.136.2 Constructor & Destructor Documentation . . . 490
6.136.2.1 UniqueAcquire(Semaphore &, bool=true) . . . 490
6.136.2.2 UniqueAcquire(const UniqueAcquire &)=delete . . . 490
6.136.2.3 UniqueAcquire(UniqueAcquire &&) . . . 490
6.136.2.4 ~UniqueAcquire() . . . 490
6.136.3 Member Function Documentation . . . 490
6.136.3.1 acquire() . . . 490
6.136.3.2 release() . . . 490
6.137 UniqueReadLock Class Reference . . . 491
6.137.1 Detailed Description . . . 491
6.137.2 Constructor & Destructor Documentation . . . 492
6.137.2.1 UniqueReadLock(const RWMutex &, bool=true) . . . 492
6.137.2.2 UniqueReadLock(const UniqueReadLock &)=delete . . . 492
6.137.2.3 UniqueReadLock(UniqueReadLock &&) . . . 493
6.137.2.4 ~UniqueReadLock() . . . 493
6.137.3 Member Function Documentation . . . 493
6.137.3.1 lock() const . . . 493
6.137.3.2 unlock() const . . . 493
6.138 UniqueWriteLock Class Reference . . . 493
6.138.1 Detailed Description . . . 494
6.138.2 Constructor & Destructor Documentation . . . 495
6.138.2.1 UniqueWriteLock(RWMutex &, bool=true) . . . 495
6.138.2.2 UniqueWriteLock(const UniqueWriteLock &)=delete . . . 495
6.138.2.3 UniqueWriteLock(UniqueWriteLock &&) . . . 496
6.138.2.4 ~UniqueWriteLock() . . . 496
6.138.3 Member Function Documentation . . . 496
6.138.3.1 lock() . . . 496

6.138.3.2 unlock() . . . 496
6.139 URLVariables Class Reference . . . 496
6.139.1 Detailed Description . . . 497
6.139.2 Constructor & Destructor Documentation . . . 497
6.139.2.1 URLVariables(const WeakString &flashVars) . . . 497
6.139.3 Member Function Documentation . . . 498
6.139.3.1 encode(const WeakString &) -> std::string . . . 498
6.139.3.2 decode(const WeakString &) -> std::string . . . 498
6.139.3.3 toString() const -> std::string . . . 498
6.140 User Class Reference . . . 498
6.140.1 Detailed Description . . . 500
6.140.2 Constructor & Destructor Documentation . . . 500
6.140.2.1 User(Server *) . . . 500
6.140.3 Member Function Documentation . . . 501
6.140.3.1 getSemaphore() const -> library::Semaphore * . . . 501
6.140.3.2 getAuthority() const -> int . . . 501
6.140.3.3 addClient(Socket *) . . . 501
6.140.3.4 eraseClient(size_t) . . . 501
6.140.3.5 goLogin(std::shared_ptr< Invoke >) . . . 502
6.140.3.6 goJoin(std::shared_ptr< Invoke >) . . . 502
6.140.3.7 goLogout() . . . 502
6.140.3.8 doLogin(std::shared_ptr< Invoke >) -> bool=0 . . . 502
6.140.3.9 doJoin(std::shared_ptr< Invoke >) -> bool=0 . . . 502
6.140.3.10 sendData(std::shared_ptr< Invoke >) override . . . 503
6.140.3.11 replyData(std::shared_ptr< Invoke >) override . . . 503
6.140.4 Member Data Documentation . . . 503
6.140.4.1 sessionID . . . 503
6.140.4.2 semaphore . . . 503
6.140.4.3 id . . . 503
6.140.4.4 authority . . . 504
6.141 WeakString Class Reference . . . 504
6.141.1 Detailed Description . . . 506
6.141.2 Constructor & Destructor Documentation . . . 507
6.141.2.1 WeakString() . . . 507
6.141.2.2 WeakString(const char *data, size_t size) . . . 507
6.141.2.3 WeakString(const char *begin, const char *end) . . . 507
6.141.2.4 WeakString(const char *data) . . . 507
6.141.2.5 WeakString(const char &ch) . . . 508
6.141.2.6 WeakString(std::initializer_list< char > &il) . . . 508
6.141.2.7 WeakString(const std::string &str) . . . 508

6.141.3 Member Function Documentation . . . 508
6.141.3.1 data() const -> const char * . . . 508
6.141.3.2 size() const -> size_t . . . 508
6.141.3.3 empty() const -> bool . . . 509
6.141.3.4 at(size_t index) const -> const char & . . . 509
6.141.3.5 operator[](size_t index) const -> const char & . . . 509
6.141.3.6 find(const WeakString &delim, size_t startIndex=NULL) const -> size_t . . . 509
6.141.3.7 rfind(const WeakString &delim, size_t endIndex=SIZE_MAX) const -> size_t . . . 509
6.141.3.8 finds(const std::vector< std::string > &delims, size_t startIndex=0) const -> IndexPair< WeakString > . . . 509
6.141.3.9 rfinds(const std::vector< std::string > &delims, size_t endIndex=SIZE_MAX) const -> IndexPair< WeakString > . . . 510
6.141.3.10 substr(size_t startIndex, size_t endIndex=SIZE_MAX) const -> WeakString . . . 510
6.141.3.11 substring(size_t startIndex, size_t size=SIZE_MAX) const -> WeakString . . . 510
6.141.3.12 between(const WeakString &start={}, const WeakString &end={}) const -> WeakString . . . 511
6.141.3.13 split(const WeakString &delim) const -> std::vector< WeakString > . . . 511
6.141.3.14 betweens(const WeakString &start={}, const WeakString &end={}) const -> std↩::vector< WeakString > . . . 511
6.141.3.15 trim(const std::vector< std::string > &delims) const -> WeakString . . . 512
6.141.3.16 ltrim(const std::vector< std::string > &delims) const -> WeakString . . . 512
6.141.3.17 rtrim(const std::vector< std::string > &delims) const -> WeakString . . . 512
6.141.3.18 replace(const WeakString &before, const WeakString &after) const -> std::string 512
6.141.3.19 replaceAll(const WeakString &before, const WeakString &after) const -> std::string 512
6.141.3.20 replaceAll(const std::vector< std::pair< std::string, std::string >> &pairs) const -> std::string . . . 513
6.141.3.21 toLowerCase() const -> std::string . . . 513
6.141.3.22 toUpperCase() const -> std::string . . . 513
6.141.3.23 str() const -> std::string . . . 513
6.141.4 Member Data Documentation . . . 513
6.141.4.1 npos . . . 513
6.142 WebServerConnector Class Reference . . . 514
6.142.1 Detailed Description . . . 514
6.143 Wrapper Class Reference . . . 515
6.143.1 Detailed Description . . . 516
6.143.2 Constructor & Destructor Documentation . . . 517
6.143.2.1 Wrapper(const std::string &name, int price, int volume, int weight) . . . 517
6.143.2.2 Wrapper(const Wrapper &wrapper) . . . 517
6.143.3 Member Function Documentation . . . 517
6.143.3.1 tryInsert(Product *) -> bool . . . 517
6.143.3.2 toString() const -> std::string override . . . 518

6.144 WrapperArray Class Reference . . . 518
6.144.1 Detailed Description . . . 519
6.144.2 Constructor & Destructor Documentation . . . 519
6.144.2.1 WrapperArray(Wrapper ∗) . . . 519
6.144.3 Member Function Documentation . . . 519
6.144.3.1 tryInsert(Product ∗) -> bool . . . 519
6.144.3.2 optimize() . . . 520
6.144.3.3 calcPrice() const -> int . . . 520
6.145 XML Class Reference . . . 520
6.145.1 Detailed Description . . . 521
6.145.2 Constructor & Destructor Documentation . . . 523
6.145.2.1 XML() . . . 523
6.145.2.2 XML(const XML &) . . . 524
6.145.2.3 XML(WeakString) . . . 524
6.145.2.4 XML(XML ∗, WeakString &) . . . 524
6.145.3 Member Function Documentation . . . 524
6.145.3.1 push_back(const WeakString &) . . . 524
6.145.3.2 push_back(const std::shared_ptr< XML >) . . . 524
6.145.3.3 addAllProperty(const std::shared_ptr< XML >) . . . 524
6.145.3.4 setTag(const std::string &) . . . 525
6.145.3.5 setValue(const _Ty &val) . . . 525
6.145.3.6 eraseProperty(const std::string &) . . . 525
6.145.3.7 getTag() const -> std::string . . . 525
6.145.3.8 toString(size_t level=0) const -> std::string . . . 525
6.145.4 Member Data Documentation . . . 526
6.145.4.1 tag . . . 526
6.145.4.2 value . . . 526
6.145.4.3 propertyMap . . . 526

Index **527**

Chapter 1

Samchon Framework

Samchon Framework is a framework for

- Providing libraries.
- Realizing cloud server(service).
- Constituting Distributed, Parallel processing system.
- Realizing composite and logical retrivalance model; Nam-Tree.

Related Homepages

- `Homepage of Samchon Framework`
- `Github - Samchon Framework`
- Examples
  - `Samchon Simulation Cloud`
  - `Hansung Timetable v3.0` (id: guest, password: N/A)

Related Documents

- Guidance for Developers
  - `English`
  - `Korean`
- Architecture Designs
  - `Class Diagram of CPP`
  - `Class Diagram of JS(TypeScript and Flex(JS))`
  - `Sequence Diagram`
  - `Entity Relationship Diagram`
- Another API Documents
  - `C++ as PDF file`
  - `Flex`
  - `TypeScript`

Author

Jeongho Nam

- E-mail: `samchon@samchon.org`
- Mobile: 82-10-3627-0016

Copyright

BSD license

Chapter 2

Namespace Index

2.1 Namespace List

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

samchon
Top level namespace of products built from samchon . . . . . ??
samchon::example::chat_service
A chat service inherited from cloud service . . . . . ??
samchon::example::console_chat_server
A chat server . . . . . ??
samchon::example::tsp
A packer with number of wrapper type and products . . . . . ??
samchon::library
Package of libraries . . . . . ??
samchon::namtree
Package of Nam-Tree . . . . . ??
samchon::protocol
Package of network protocol and libraries . . . . . ??
samchon::protocol::master
Package for external system, within the framework of master . . . . . ??
samchon::protocol::service
Package of cloud service as a server . . . . . ??
samchon::protocol::slave
Package of external system, within the framework of slave . . . . . ??

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Base64 . . . 63
- ByteArray . . . 65
- CaseGenerator . . . 69
  - CombinedPermutationGenerator . . . 105
  - PermutationGenerator . . . 424
    - FactorialGenerator . . . 247
- Charset . . . 79
- CriticalAllocator< _Ty > . . . 107
- Date . . . 111
  - Datetime . . . 119
- Entity . . . 183
  - EntityGroup< _Container, Entity * > . . . 195
    - IEntityPtrGroup< _Container > . . . 288
  - EntityGroup< _Container, std::shared_ptr< Entity > > . . . 195
  - EntityGroup< _Container, std::unique_ptr< Entity > > . . . 195
  - ChatMessage . . . 86
  - ChatRoom . . . 88
  - FTInstance . . . 259
    - FTFile . . . 254
      - FTByteFile . . . 250
      - FTTextFile . . . 262
      - NTFile . . . 357
    - FTFolder . . . 256
  - INTExplore . . . 296
    - NTFile . . . 357
    - NTParameter . . . 363
  - NTParameterDetermined . . . 369
  - NTSide . . . 372
  - EntityArray< _Ty > . . . 189
  - EntityGroup< _Container, _ETy, _Ty > . . . 195
    - FTFolder . . . 256
    - NTCriteria . . . 348
    - NTParameter . . . 363
    - NTParameterArray . . . 366
    - ExternalSystem . . . 235
      - ExternalClient . . . 223

DistributedClient . . . 128
ParallelClient . . . 379
SlaveClient . . . 456
DistributedClientServerMediator . . . 140
ParallelClient . . . 377
ParallelSlaveClientMediator . . . 400
ExternalServer . . . 229
DistributedServer . . . 143
ParallelServer . . . 391
SlaveServer . . . 458
DistributedSlaveServerMediator . . . 153
ParallelServer . . . 389
ParallelSlaveServerMediator . . . 404
DistributedSystem . . . 160
DistributedClient . . . 128
DistributedServer . . . 143
ParallelSystem . . . 410
ParallelClient . . . 379
ParallelServer . . . 391
SlaveSystem . . . 460
DistributedSlaveSystemMediator . . . 157
DistributedClientServerMediator . . . 140
DistributedSlaveServerMediator . . . 153
ParallelSystem . . . 414
ParallelSlaveSystemMediator . . . 407
ParallelSlaveClientMediator . . . 400
ParallelSlaveServerMediator . . . 404
ParallelClient . . . 377
ParallelServer . . . 389
SlaveClient . . . 456
SlaveServer . . . 458
ExternalSystemArray . . . 239
ExternalClientArray . . . 225
DistributedClientArray . . . 130
DistributedClientArrayMediator . . . 135
ParallelClientArray . . . 380
ParallelClientArrayMediator . . . 383
ExternalServerArray . . . 232
DistributedServerArray . . . 146
DistributedServerArrayMediator . . . 149
ParallelServerArray . . . 393
ParallelServerArrayMediator . . . 396
DistributedSystemArray . . . 165
DistributedClientArray . . . 130
DistributedServerArray . . . 146
DistributedSystemArrayMediator . . . 170
DistributedClientArrayMediator . . . 135
DistributedServerArrayMediator . . . 149
ParallelSystemArray . . . 417
ParallelClientArray . . . 380
ParallelServerArray . . . 393
ParallelSystemArrayMediator . . . 419
ParallelClientArrayMediator . . . 383
ParallelServerArrayMediator . . . 396
Invoke . . . 300
InvokeHistoryArray . . . 310

PRInvokeHistoryArray . . . 429
DSInvokeHistoryList . . . 181
EntityList< _Ty > . . . 206
ExternalSystemRole . . . 242
DistributedSystemRole . . . 176
InvokeHistory . . . 306
DSInvokeHistory . . . 179
PRInvokeHistory . . . 427
InvokeParameter . . . 312
SystemRole . . . 480
Event . . . 216
ErrorEvent . . . 213
ProgressEvent . . . 432
EventDispatcher . . . 219
FlashPolicyServer . . . 248
FTFactory . . . 251
NTFactory . . . 354
GAParameters . . . 264
GAParameters . . . 265
GAPopulation< GeneArray, Compare > . . . 266
GeneticAlgorithm< GeneArray, Compare > . . . 269
GeometryPoint . . . 278
HTTPLoader . . . 281
IEntityChain . . . 285
IEntityGroup . . . 287
EntityGroup< _Container, Entity * > . . . 195
EntityGroup< _Container, std::shared_ptr< Entity > > . . . 195
EntityGroup< _Container, std::unique_ptr< Entity > > . . . 195
ChatRoomArray . . . 91
EntityArray< _Ty > . . . 189
EntityGroup< _Container, _ETy, _Ty > . . . 195
EntityList< _Ty > . . . 206
IHTMLEntity . . . 288
IndexPair< _Ty > . . . 294
Instance . . . 295
Product . . . 431
Wrapper . . . 515
IOperator< _Ty > . . . 317
IOperator< NTIterator > . . . 317
NTIterator . . . 361
IProtocol . . . 318
ChatRoom . . . 88
ChatServer . . . 94
ExternalSystemArray . . . 239
ExternalSystemRole . . . 242
IClient . . . 282
ChatClient . . . 84
ExternalSystem . . . 235
IWebClient . . . 328
WebServerConnector . . . 514
ServerConnector . . . 442
ExternalServer . . . 229
WebServerConnector . . . 514
Client . . . 100
ChatClient . . . 82
Service . . . 444

ChatService . . . 97
ListService . . . 335
User . . . 498
ChatUser . . . 98
SystemRole . . . 480
IPUserPair . . . 320
IServer . . . 322
ChatServer . . . 94
ExternalClientArray . . . 225
IWebServer . . . 331
Server . . . 440
ChatServer . . . 95
ISQLEntity . . . 327
Map< _Kty, _Ty, _Pr, _Alloc > . . . 336
Server . . . 440
Map< int, samchon::library::FTInstance * > . . . 336
Map< SideFunction > . . . 336
Map< size_t, SmartPointer< Client > > . . . 336
User . . . 498
Map< size_t, SQL_SIZE_T > . . . 336
Map< std::shared_ptr< samchon::protocol::master::DistributedSystemRole > > . . . 336
Map< std::shared_ptr< samchon::protocol::service::IPUserPair > > . . . 336
Map< std::string, std::shared_ptr< XMLList > > . . . 336
XML . . . 520
Map< std::string, std::string > . . . 336
URLVariables . . . 496
Math . . . 339
NTEntityGroup . . . 352
Packer . . . 375
RWMutex . . . 434
Scheduler . . . 436
Semaphore . . . 437
ServiceKeeper . . . 446
SharedAcquire . . . 447
SharedReadLock . . . 450
SharedWriteLock . . . 453
SmartPointer< _Ty > . . . 462
SmartPointer< samchon::protocol::service::Client > . . . 462
SmartPointer< samchon::protocol::service::User > . . . 462
SQLi . . . 465
MySQLi . . . 346
TSQLi . . . 484
SQLStatement . . . 467
TSQLStatement . . . 486
StringUtil . . . 472
Travel . . . 482
UniqueAcquire . . . 488
UniqueReadLock . . . 491
UniqueWriteLock . . . 493
WeakString . . . 504
WrapperArray . . . 518

Chapter 4

Class Index

4.1 Class List

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

Base64
Utility class for base64 format's en-decoding . . . . . ??
ByteArray
Binary data class . . . . . ??
CaseGenerator
Case generator . . . . . ??
Charset
A utility class supporting conversion between multiple character-sets . . . . . ??
ChatClient
A client . . . . . ??
ChatClient
A chat client . . . . . ??
ChatMessage
A chat message . . . . . ??
ChatRoom
A chat room . . . . . ??
ChatRoomArray
A manager of chat rooms . . . . . ??
ChatServer
A chat server . . . . . ??
ChatServer
A chat server . . . . . ??
ChatService
A chat service . . . . . ??
ChatUser
An user in chat server . . . . . ??
Client
A network boundary with the client in an User . . . . . ??
CombinedPermutationGenerator
A combined-permutation case generator . . . . . ??
CriticalAllocator< _Ty >
An allocator ensuring concurrency . . . . . ??
Date
Date (year, month, day in month) . . . . . ??
Datetime
Date and time . . . . . ??
DistributedClient
A network driver for a distributed client . . . . . ??

DistributedClientArray
An array of DistributedClient(s) . . . . . ??
DistributedClientArrayMediator
A mediator of distributed system between master and slaves . . . . . ??
DistributedClientServerMediator
A distributed slave client for mediation . . . . . ??
DistributedServer
A network driver for a distributed server . . . . . ??
DistributedServerArray
An array of DistributedServer(s) . . . . . ??
DistributedServerArrayMediator
A mediator of distributed system between master and slaves . . . . . ??
DistributedSlaveServerMediator
A distributed slave server for mediation . . . . . ??
DistributedSlaveSystemMediator
A distributed slave system for mediation . . . . . ??
DistributedSystem
A network driver for a distributed system . . . . . ??
DistributedSystemArray
An array of distributed system drivers . . . . . ??
DistributedSystemArrayMediator
A mediator of distributed system between master and slaves . . . . . ??
DistributedSystemRole
A role of distributed processing system . . . . . ??
DSInvokeHistory
A reported history of an Invoke message . . . . . ??
DSInvokeHistoryList
A list of DSInvokeHistory . . . . . ??
Entity
An entity, a standard data class . . . . . ??
EntityArray< _Ty >
An Entity and a static array containing Entity objects . . . . . ??
EntityGroup< _Container, _ETy, _Ty >
An Entity and a container of children Entity objects . . . . . ??
EntityList< _Ty >
An Entity and a static list containing Entity objects . . . . . ??
ErrorEvent
Event representing an error . . . . . ??
Event
Represent an event running on background . . . . . ??
EventDispatcher
Abstract class for dispatching Event . . . . . ??
ExternalClient
A network driver for an external client . . . . . ??
ExternalClientArray
An array of ExternalClient(s) . . . . . ??
ExternalServer
A network driver for an external server . . . . . ??
ExternalServerArray
An array of ExternalServer(s) . . . . . ??
ExternalSystem
A network driver for an external system . . . . . ??
ExternalSystemArray
An array of ExternalSystem(s) . . . . . ??
ExternalSystemRole
A role belongs to an external system . . . . . ??
FactorialGenerator
Factorical case generator . . . . . ??

FlashPolicyServer
A flash policy server . . . . . ??
FTByteFile
A binary file . . . . . ??
FTFactory
A Factory and manager for file-tree . . . . . ??
FTFile
A file . . . . . ??
FTFolder
A folder . . . . . ??
FTInstance
An interface of file-tree . . . . . ??
FTTextFile
A text file . . . . . ??
GAParameters
A structure containing parameters of genetic algorithm . . . . . ??
GAParameters
Parameters for Genetic-Algorithm . . . . . ??
GAPopulation< GeneArray, Compare >
A population of a generation in G.A . . . . . ??
GeneticAlgorithm< GeneArray, Compare >
A genetic algorithm class . . . . . ??
GeometryPoint
A geometry coordinates (point) . . . . . ??
HTTPLoader
A http, web-page loader . . . . . ??
IClient
An interface for a client . . . . . ??
IEntityChain
A chain of entity . . . . . ??
IEntityGroup
An iternface for entity group . . . . . ??
IEntityPtrGroup< _Container > . . . . . ??
IHTMLEntity
An interface supporting conversion to html . . . . . ??
IndexPair< _Ty >
A pair of index and its value(_Ty) . . . . . ??
Instance
A physical instance . . . . . ??
INTExplore
An interface for exploration . . . . . ??
Invoke
Standard message of network I/O . . . . . ??
InvokeHistory
A history of an Invoke message . . . . . ??
InvokeHistoryArray
An array of InvokeHistory . . . . . ??
InvokeParameter
A parameter of an Invoke . . . . . ??
IOperator< _Ty >
Interface for comparision operator . . . . . ??
IProtocol
An interface of Invoke message chain . . . . . ??
IPUserPair
Relationship between IP address and User for issuing session . . . . . ??
IServer
An interface of a physical server . . . . . ??

ISQLEntity
An interface supporting DB-I/O . . . . . ??
IWebClient
An interface for a web-client . . . . . ??
IWebServer
An interface for a physical server following web-socket . . . . . ??
ListService
A list service of chat rooms . . . . . ??
Map< _Kty, _Ty, _Pr, _Alloc >
Map . . . . . ??
Math
Utility of Math . . . . . ??
MySQLi
A SQL interface for My-SQL . . . . . ??
NTCriteria
Criteria, a conditional expression with weight . . . . . ??
NTEntityGroup
A historical, studying data . . . . . ??
NTFactory
A factory for Nam-Tree objects . . . . . ??
NTFile
A file archiving metadata of a function . . . . . ??
NTIterator
Iterator of historical data . . . . . ??
NTParameter
A metadata of a parameter in a function . . . . . ??
NTParameterArray
An Array of NTParameter . . . . . ??
NTParameterDetermined
A pre-determined value of a parameter . . . . . ??
NTSide
A side of a conditional expresson . . . . . ??
Packer
A packer planning the best packaging . . . . . ??
ParallelClient
A slave parallel client . . . . . ??
ParallelClient
A network driver for a parallel client . . . . . ??
ParallelClientArray
An array of ParallelClient(s) . . . . . ??
ParallelClientArrayMediator
A mediator of parallel system between master and slaves . . . . . ??
ParallelServer
A slave parallel server . . . . . ??
ParallelServer
A network driver for a parallel server . . . . . ??
ParallelServerArray
An array of ParallelServer(s) . . . . . ??
ParallelServerArrayMediator
A mediator of parallel system between master and slaves . . . . . ??
ParallelSlaveClientMediator
A parallel slave client for mediation . . . . . ??
ParallelSlaveServerMediator
A parallel slave server for mediation . . . . . ??
ParallelSlaveSystemMediator
A parallel slave system for mediation . . . . . ??
ParallelSystem
A network driver for a parallel system . . . . . ??

ParallelSystem
A slave parallel system . . . . . ??
ParallelSystemArray
An array of parallel system drivers . . . . . ??
ParallelSystemArrayMediator
A mediator of parallel system between master and slaves . . . . . ??
PermutationGenerator
A permutation case generator . . . . . ??
PRInvokeHistory
A reported history of an Invoke message . . . . . ??
PRInvokeHistoryArray
An array of PRInvokeHistory . . . . . ??
Product
A product, merchandise . . . . . ??
ProgressEvent
Event representing a progress . . . . . ??
RWMutex
Rw_mutex . . . . . ??
Scheduler
A scheduler . . . . . ??
Semaphore
A semaphore . . . . . ??
Server
A server for (cloud) service . . . . . ??
ServerConnector
A server connector for a physical client . . . . . ??
Service
A service . . . . . ??
ServiceKeeper
A keeper blocking destrunction of User and Client . . . . . ??
SharedAcquire
Shared acquire from a Semaphore . . . . . ??
SharedReadLock
Shared lock from a RWMutex . . . . . ??
SharedWriteLock
Shared lock from a RWMutex . . . . . ??
SlaveClient
A slave client . . . . . ??
SlaveServer
A slave server . . . . . ??
SlaveSystem
A slave system . . . . . ??
SmartPointer< _Ty >
Global shared pointer
??
SQLi
A SQL interface; DBMS connector . . . . . ??
SQLStatement
A sql statement . . . . . ??
StringUtil
Utility class for string . . . . . ??
SystemRole
A role belongs to a system . . . . . ??
Travel
Represent a travel containning Point(s) . . . . . ??
TSQLi
A SQL interface for T-SQL . . . . . ??

TSQLStatement
A t-sql statement . . . . . ??
UniqueAcquire
Unique acquire from a Semaphore . . . . . ??
UniqueReadLock
Unique lock for reading . . . . . ??
UniqueWriteLock
Unique lock for writing . . . . . ??
URLVariables
URLVariables class is for representing variables of HTTP . . . . . ??
User
User containing Client(s) with session-id . . . . . ??
WeakString
A string class only references characeters, reference only . . . . . ??
WebServerConnector
A web-socket server connector . . . . . ??
Wrapper
A wrapper can contain products . . . . . ??
WrapperArray
An array of wrapper with same category (name) . . . . . ??
XML
XML is a class representing xml object . . . . . ??

Chapter 5

Namespace Documentation

5.1 samchon Namespace Reference

Top level namespace of products built from samchon.

Namespaces

- library

  Package of libraries.
- namtree

  Package of Nam-Tree.
- protocol

  Package of network protocol and libraries.

Classes

- class ByteArray

  Binary data class.
- class IndexPair

  A pair of index and its value(_Ty)
- class Map

  Map.
- class SmartPointer

  Global shared pointer

  .
- class WeakString

  A string class only references characeters, reference only.

Typedefs

- template<typename _Ty , typename _Pr = std::less<std::string>, typename _Alloc = std::allocator<std::pair<const std::string, _↩ Ty>>>

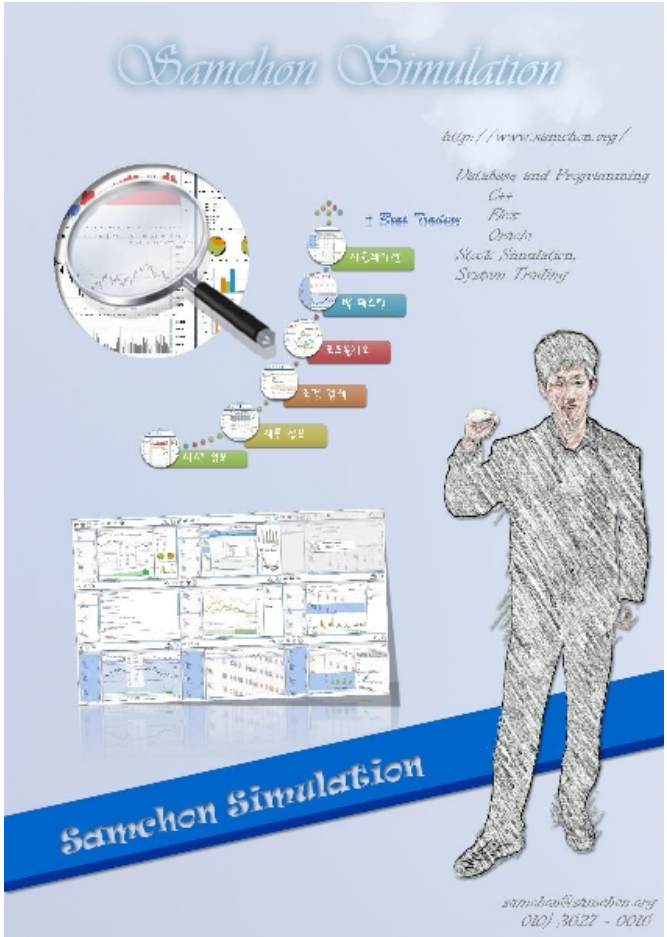
  using Dictionary = Map< std::string, _Ty, _Pr, _Alloc >

  A Map key is string.

5.1.1 Detailed Description

Top level namespace of products built from samchon.

- http://samchon.org/
- https://github.com/samchon/



Author

Jeongho Nam

5.1.2 Typedef Documentation

5.1.2.1 using Dictionary = Map<std::string, _Ty, _Pr, _Alloc>

A Map key is string.

See also

library::Map

Author

Jeongho Nam

5.2 samchon::example::chat_service Namespace Reference

A chat service inherited from cloud service.

Classes

- class ChatClient

  A client.

- class ChatMessage

  A chat message.

- class ChatRoom

  A chat room.

- class ChatRoomArray

  A manager of chat rooms.

- class ChatServer

  A chat server.

- class ChatService

  A chat service.

- class ChatUser

  An user in chat server.

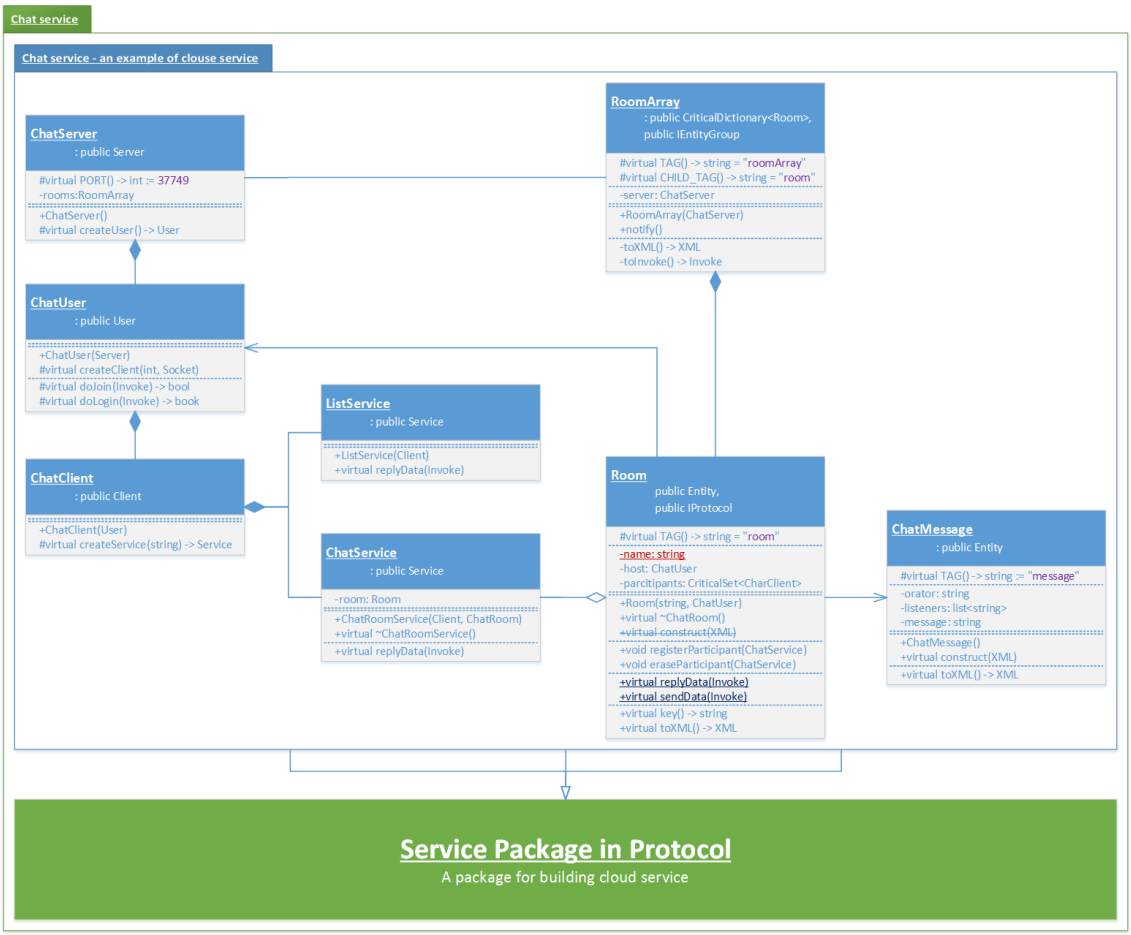
- class ListService

  A list service of chat rooms.

5.2.1 Detailed Description

A chat service inherited from cloud service.

An example for realizing cloud service and using critical allocators.



Author

Jeongho Nam

5.3 samchon::example::console_chat_server Namespace Reference

A chat server.

Classes

- class ChatClient

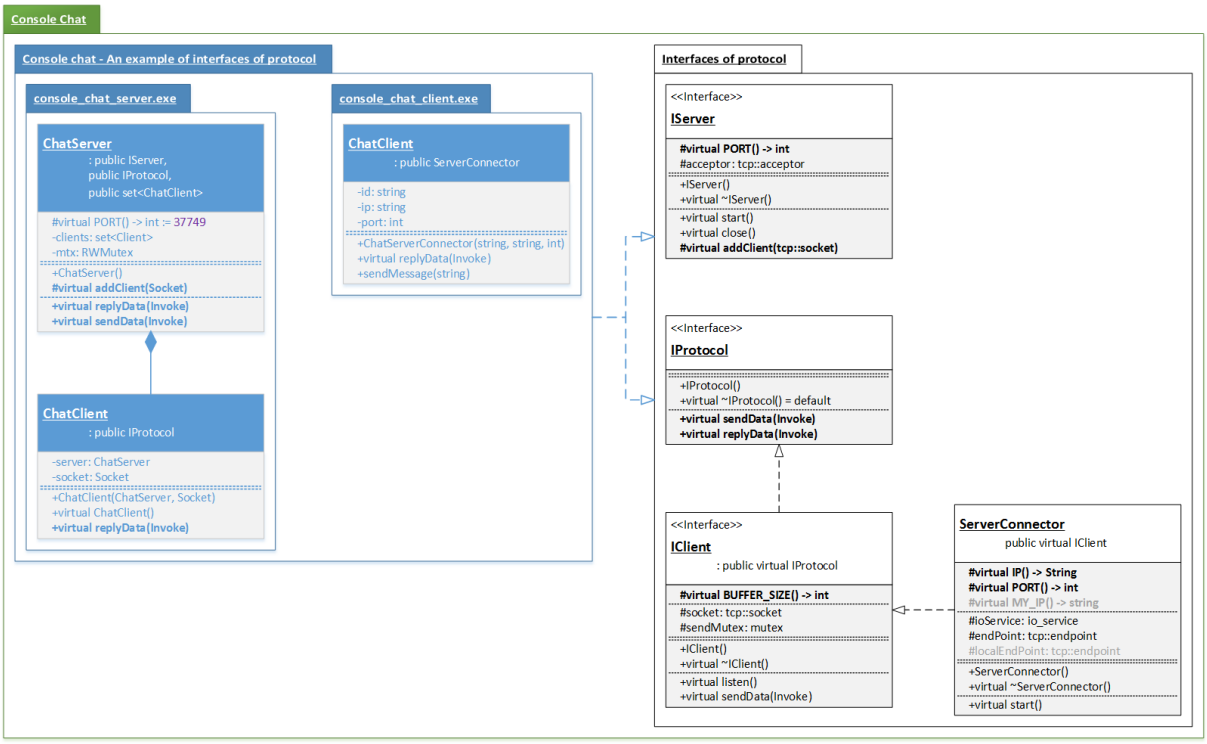
  A chat client.
- class ChatServer

  A chat server.

5.3.1 Detailed Description

A chat server.

A simple chat server operates on a console.



Author

Jeongho Nam

5.4 samchon::example::tsp Namespace Reference

A packer with number of wrapper type and products.

Classes

- struct GAParameters

  Parameters for Genetic-Algorithm.
- class GeometryPoint

  A geometry coordinates (point)
- class Scheduler

  A scheduler.
- class Travel

  Represent a travel containning Point(s)

5.4.1 Detailed Description

A packer with number of wrapper type and products.

Traveling Salesman Problem Solver.

Retrieves the solution of packaging by combination permuation and factorial case.

```
D:\OneDrive\Project\Samchon\framework\cpp\x64\Release\packer.exe
$480
Category - Large
        Wrapper Large: $100, 200cm^3, 1000g
                Product Umbrella: $4000, 200cm^3, 1000g
        Wrapper Large: $100, 200cm^3, 1000g
                Product Eraser: $500, 10cm^3, 70g
                Product Pencil: $400, 30cm^3, 35g
                Product Notebook-PC: $800000, 150cm^3, 850g

Category - Medium
        Wrapper Medium: $70, 150cm^3, 500g
                Product Drink: $1000, 75cm^3, 250g
                Product Pencil: $400, 30cm^3, 35g
                Product Pencil: $400, 30cm^3, 35g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Book: $8000, 150cm^3, 300g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Book: $8000, 150cm^3, 300g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Tablet-PC: $600000, 120cm^3, 450g

Category - Small


Press any key to continue . . .
```


```
D:\OneDrive\Project\Samchon\framework\cpp\x64\Release\packer.exe
$480
Category - Large
        Wrapper Large: $100, 200cm^3, 1000g
                Product Umbrella: $4000, 200cm^3, 1000g
        Wrapper Large: $100, 200cm^3, 1000g
                Product Eraser: $500, 10cm^3, 70g
                Product Pencil: $400, 30cm^3, 35g
                Product Notebook-PC: $800000, 150cm^3, 850g

Category - Medium
        Wrapper Medium: $70, 150cm^3, 500g
                Product Drink: $1000, 75cm^3, 250g
                Product Pencil: $400, 30cm^3, 35g
                Product Pencil: $400, 30cm^3, 35g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Book: $8000, 150cm^3, 300g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Book: $8000, 150cm^3, 300g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Tablet-PC: $600000, 120cm^3, 450g

Category - Small


Press any key to continue . . .
```


See also

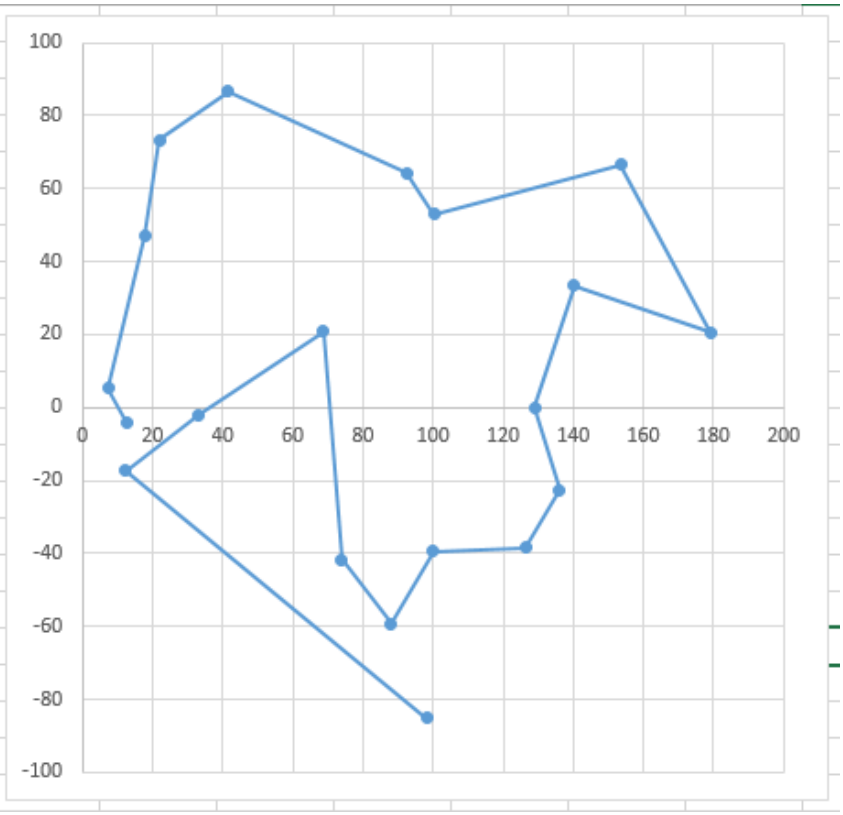
library::CaseGenerator

Author

Jeongho Nam

Deducts an optimal traveling scheduel by genetic algorithm.

Distance: 64,896.70 km		
uid	longitude	latitude
18	12.5233	-3.95529
15	7.23275	5.18342
7	17.7528	46.8665
2	22.0434	73.4133
20	41.6615	86.5372
13	92.427	64.3369
1	100.345	53.1067
9	153.521	66.5119
3	179.118	20.5073
10	140.113	33.3537
17	128.885	0.054607
4	136.012	-22.4915
6	126.352	-38.5031
16	99.9775	-39.3696
11	87.8859	-59.3663
14	73.8477	-41.5867
5	68.6563	20.8528
12	33.2162	-1.92823
8	12.1839	-17.3552
19	98.0967	-85.195



cpp/resultexample_tsp.png

Author

Jeongho Nam

5.5 samchon::library Namespace Reference

Package of libraries.

Classes

- class Base64

  Utility class for base64 format's en-decoding.
- class CaseGenerator

  Case generator.
- class Charset

  A utility class supporting conversion between multiple character-sets.
- class CombinedPermutationGenerator

  A combined-permutation case generator.
- class CriticalAllocator

  An allocator ensuring concurrency.
- class Date

  Date (year, month, day in month)
- class Datetime

Date and time.

- class ErrorEvent

  Event representing an error.
- class Event

  Represent an event running on background.
- class EventDispatcher

  Abstract class for dispatching Event.
- class FactorialGenerator

  Factorical case generator.
- class FTByteFile

  A binary file.
- class FTFactory

  A Factory and manager for file-tree.
- class FTFile

  A file.
- class FTFolder

  A folder.
- class FTInstance

  An interface of file-tree.
- class FTTextFile

  A text file.
- struct GAParameters

  A structure containing parameters of genetic algorithm.
- class GAPopulation

  A population of a generation in G.A.
- class GeneticAlgorithm

  A genetic algorithm class.
- class HTTPLoader

  A http, web-page loader.
- class IOperator

  Interface for comparision operator.
- class Math

  Utility of Math.
- class MySQLi

  A SQL interface for My-SQL.
- class PermutationGenerator

  A permutation case generator.
- class ProgressEvent

  Event representing a progress.
- class RWMutex

  rw_mutex
- class Semaphore

  A semaphore.
- class SharedAcquire

  Shared acquire from a Semaphore.
- class SharedReadLock

  Shared lock from a RWMutex.
- class SharedWriteLock

  Shared lock from a RWMutex.
- class SQLi

  A SQL interface; DBMS connector.

- class SQLStatement

  A sql statement.
- class StringUtil

  Utility class for string.
- class TSQLi

  A SQL interface for T-SQL.
- class TSQLStatement

  A t-sql statement.
- class UniqueAcquire

  Unique acquire from a Semaphore.
- class UniqueReadLock

  Unique lock for reading.
- class UniqueWriteLock

  Unique lock for writing.
- class URLVariables

  URLVariables class is for representing variables of HTTP.
- class XML

  XML is a class representing xml object.

Typedefs

- template<typename _Ty , typename _Pr = std::less<std::string>>
  using CriticalDictionary = CriticalMap< std::string, _Ty, _Pr >

  A Dictionary ensures concurrency.
- template<typename _Ty >
  using CriticalList = std::list< _Ty, CriticalAllocator< _Ty >>

  A std::list ensures concurrency.
- template<typename _Kty , typename _Ty , typename _Pr = std::less<_Kty>>
  using CriticalMap = Map< _Kty, _Ty, _Pr, CriticalAllocator< std::pair< const std::string, _Ty >>>

  A Map ensures concurrency.
- template<typename _Ty , typename _Pr = std::less<_Ty>>
  using CriticalSet = Set< _Ty, _Pr, CriticalAllocator< _Ty >>

  A Set ensures concurrency.
- template<typename _Ty >
  using CriticalVector = std::vector< _Ty, CriticalAllocator< _Ty >>

  A std::vector ensures concurrency.
- typedef std::vector< std::shared_ptr< XML > > XMLList

  A list of XML, tags are same.

5.5.1 Detailed Description

Package of libraries.

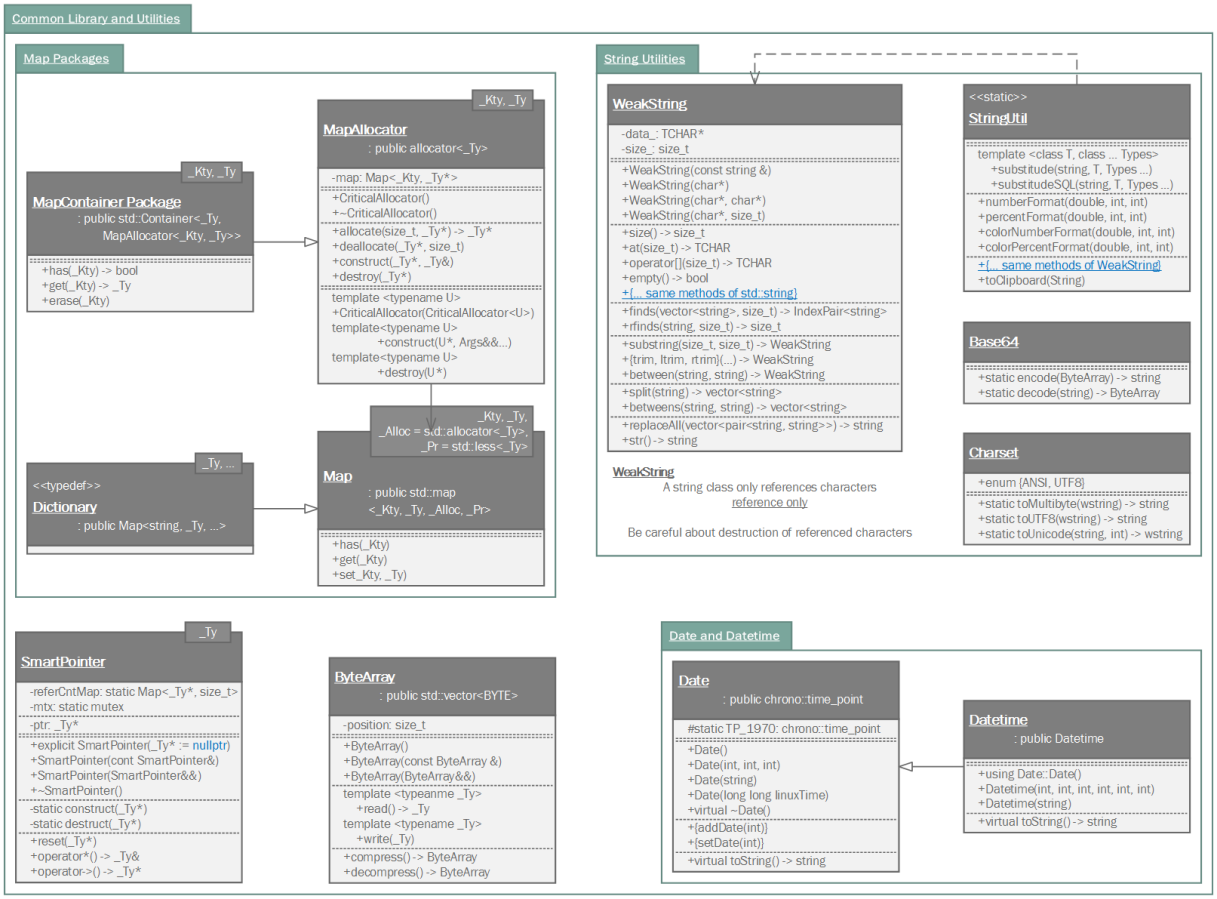
The library package supports some classes for package of protocol and nam-tree.

Furthermore, library package provides some utility libraries, which are not being supported by standard library (STL). Those can be helpful for users and those are like STL, independent from operating systems, so that can compile in any operating system.

The library package can be divided into some sectinos.

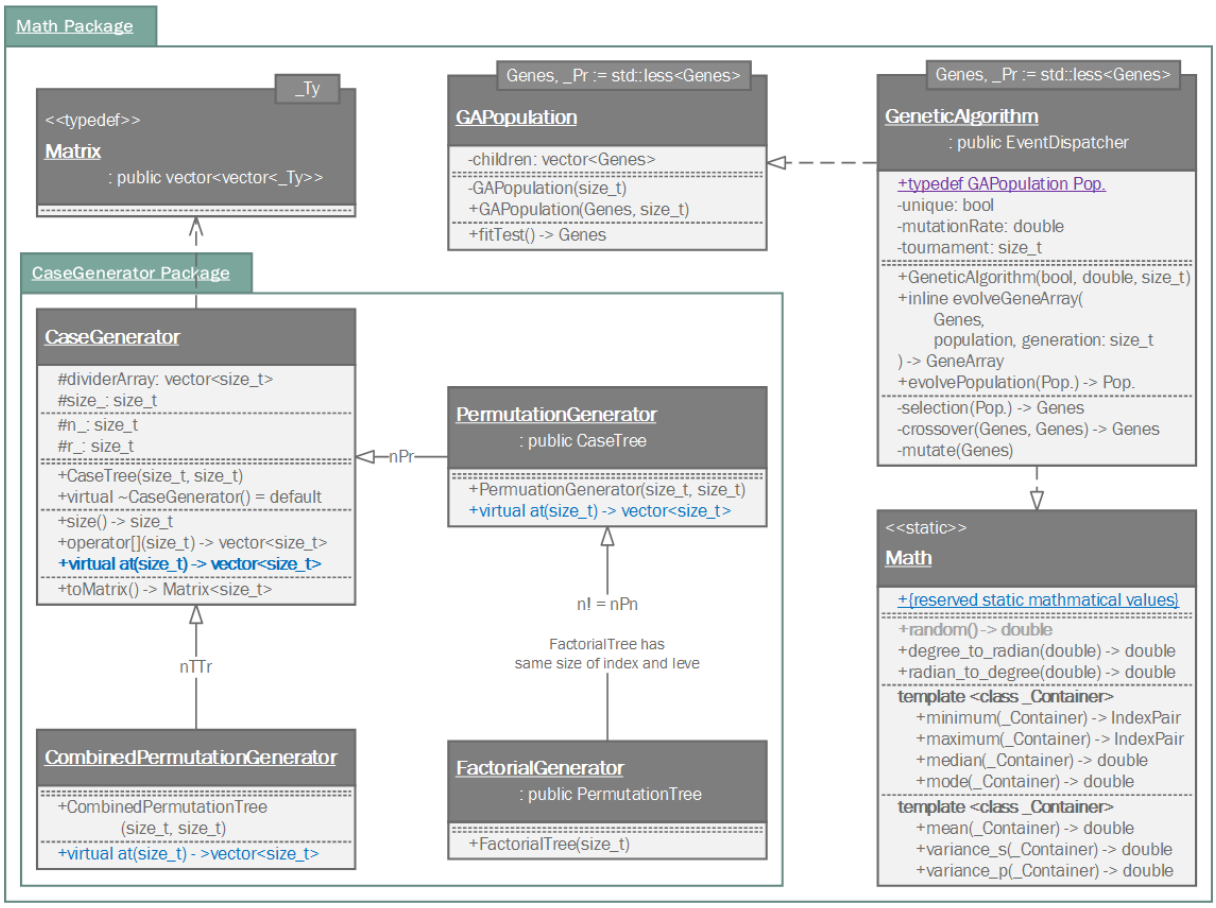
- **Common libraries.**

  Common containers with utility methods.



- **Mathmatical libaries.**

  Utilty classes for mathmatical methods, or libraries of case generator and genetic algorithm. In those mathmatical part, lots of examples are supported.

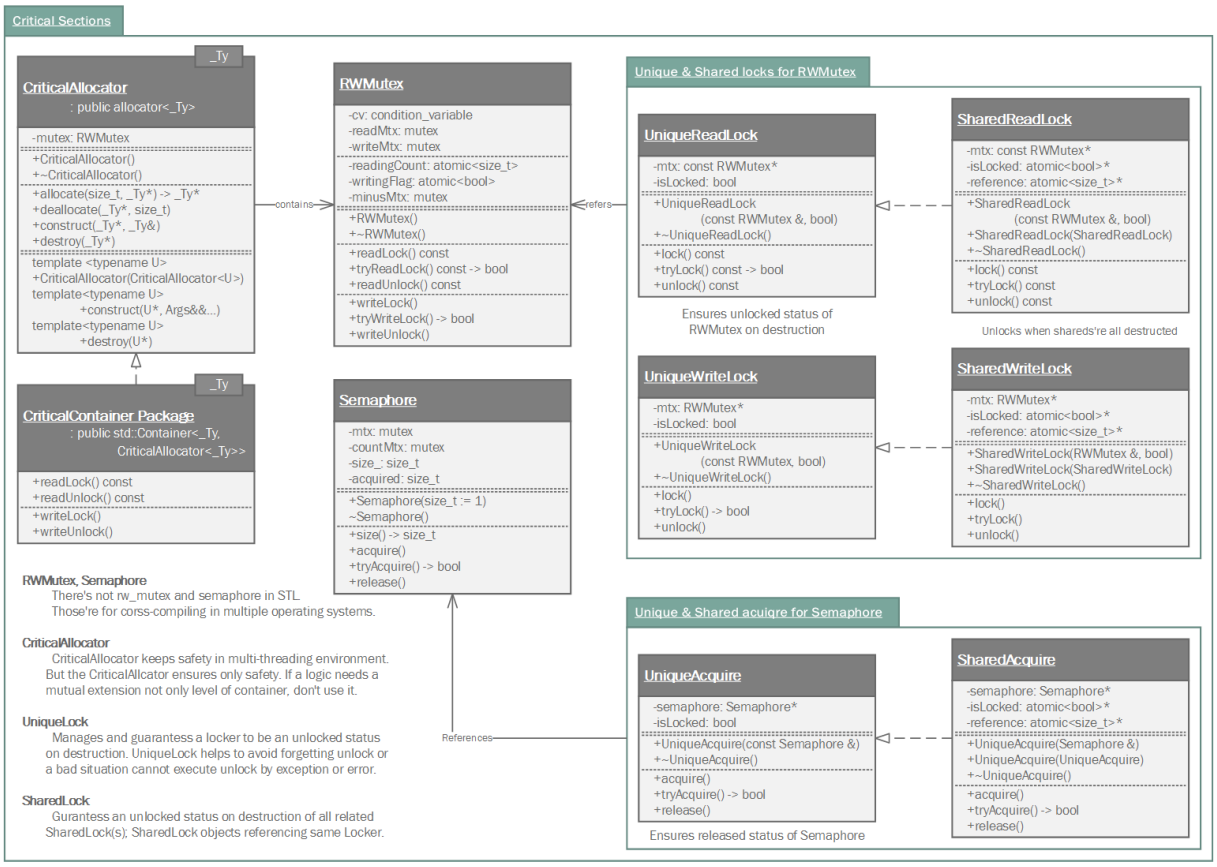


- **Critical section libraries.**

Classes handling critical section, like RWMutex and Semaphore which are not supported in STL yet.

Of course, rw_mutex is already defined in linux C, semaphore is already defined in MFC. But they are dependent on their own operating system, so that cannot compile in multiple operating systems. It's the reason why Samchon Framework provides those classes.

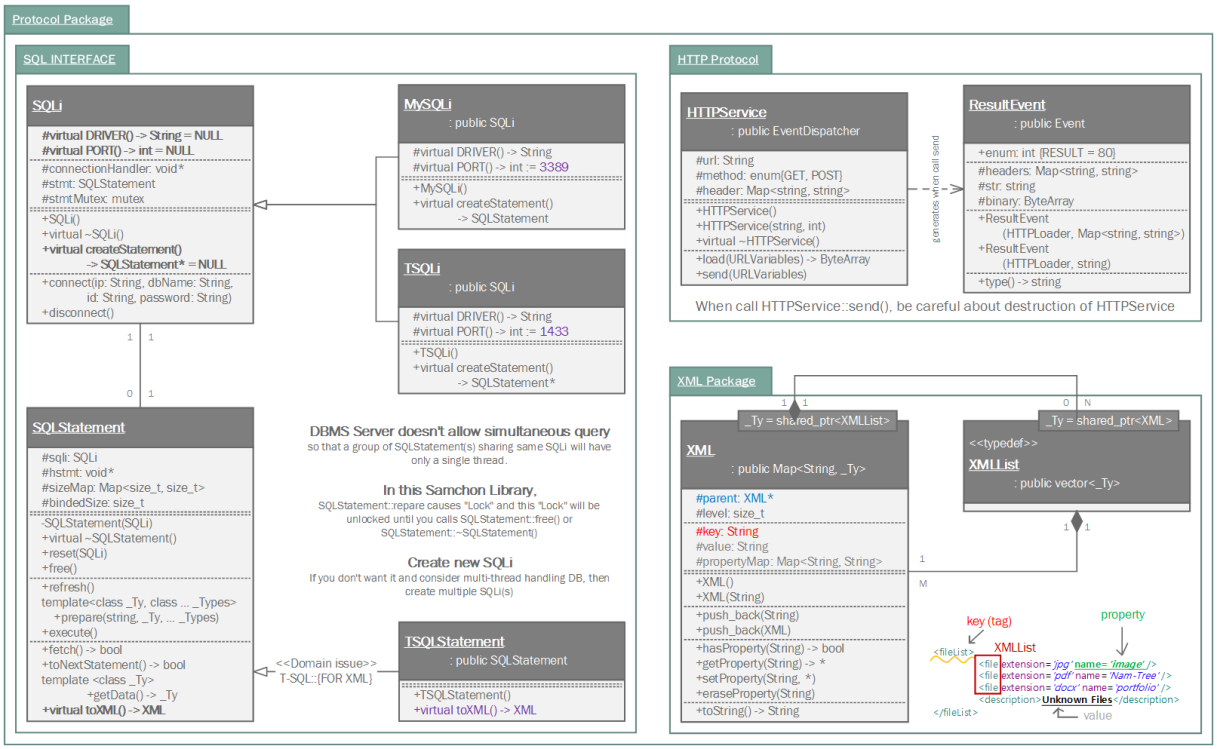
Furthermore, unique and shared lock of those critical section classes, which are managing locking and unlocking, and critical allocator are provided.



- **Data I/O libraries.**

  Provides libraries about data I/O.

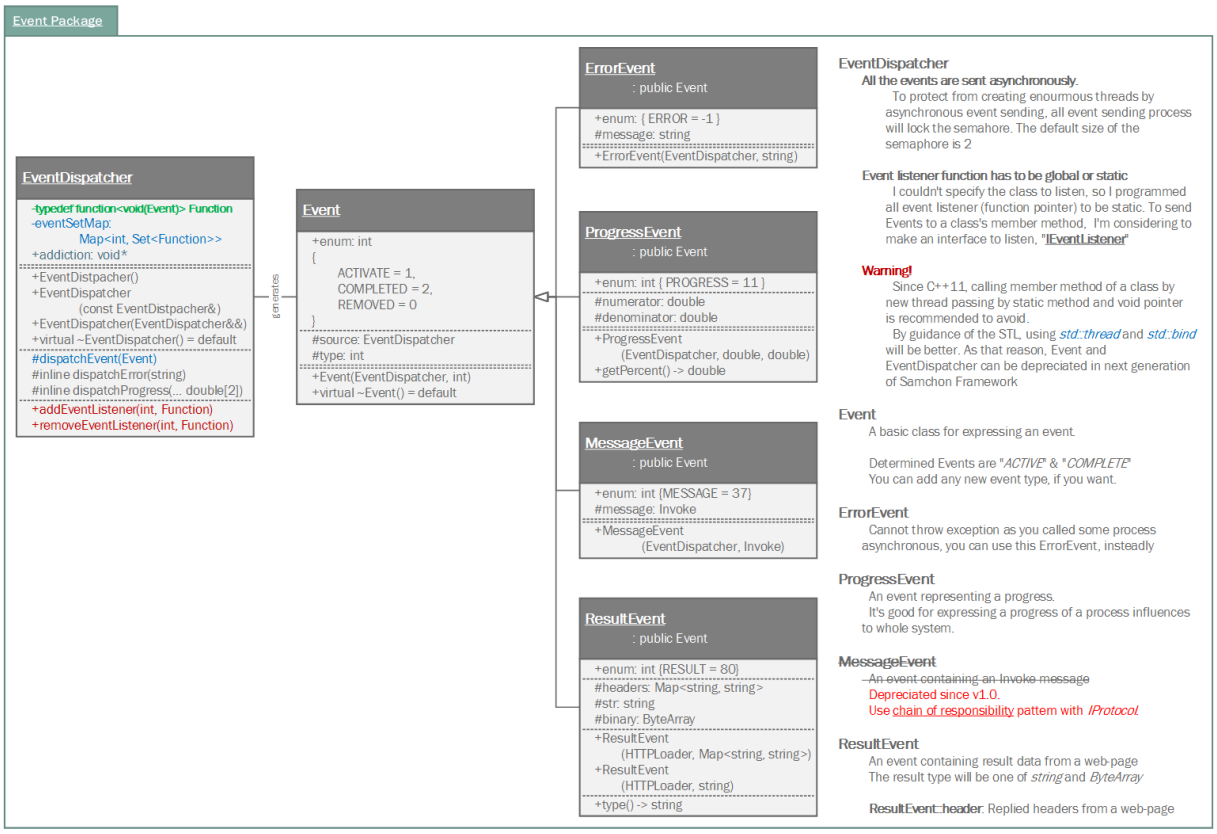
  SQLi and SQLStatement are the ODBC drivier designed to follow principles of OOP by adapter pattern. XML class is designed to follow composite relationship.



- **Event libraries.**

Libraries representing events and dispatching those events, which are running on background, with own exclusive thread.

But there's something to notice. Since C++11, calling member method of a class by new thread passing by static method and void pointer is recommended to avoid. By guidance of the STL, using std::thread and std::bind will be better. As that reason, Event and EventDispatcher can be depreciated in next generation of Samchon Framework.

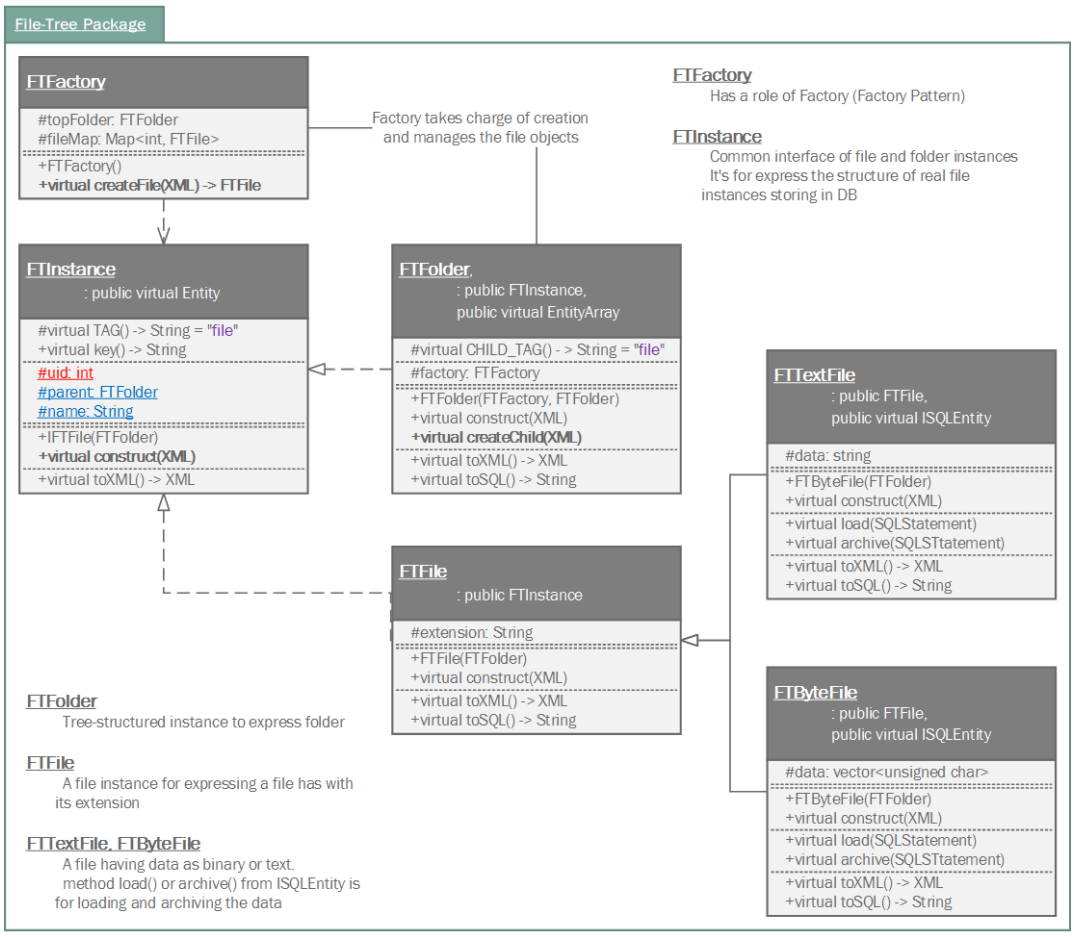


- **File-tree libraries.**

A module for expressing folder and file instances. The objects in file-tree module are realized by protocol::Entity.

They have recursive and hierarchical relationship and created by Factory class (FTFactory, factory pattern). Those objects can be archived in and loaded from Database. Using the pre-defined methods interaction with Databse, you can not only express real file and folder instances but also realize virtual file system.

Classes in the Module of file-tree are all abstract, but package nam-tree can be an example inheriting and having real model from those abstract classes.



Author

Jeongho Nam

5.5.2 Typedef Documentation

5.5.2.1 using CriticalDictionary = CriticalMap<std::string, _Ty, _Pr>

A Dictionary ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.2 using CriticalList = std::list<_Ty, CriticalAllocator<_Ty>>

A std::list ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.3 using CriticalMap = Map<_Kty, _Ty, _Pr, CriticalAllocator<std::pair<const std::string, _Ty>>>

A Map ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.4 using CriticalSet = Set<_Ty, _Pr, CriticalAllocator<_Ty>>

A Set ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.5 using CriticalVector = std::vector<_Ty, CriticalAllocator<_Ty>>

A std::vector ensures concurrency.

See also

library::CriticalAllocator
samchon::library

Author

Jeongho Nam

5.5.2.6 typedef std::vector<std::shared_ptr<XML> > XMLList

A list of XML, tags are same.

See also

samchon::library

Author

Jeongho Nam

5.6 samchon::namtree Namespace Reference

Package of Nam-Tree.

Classes

- class INTExplore

  An interface for exploration.

- class NTCriteria

  Criteria, a conditional expression with weight.

- class NTEntityGroup

  A historical, studying data.

- class NTFactory

  A factory for Nam-Tree objects.

- class NTFile

  A file archiving metadata of a function.

- class NTIterator

  Iterator of historical data.

- class NTParameter

  A metadata of a parameter in a function.

- class NTParameterArray

  An Array of NTParameter.

- class NTParameterDetermined

  A pre-determined value of a parameter.

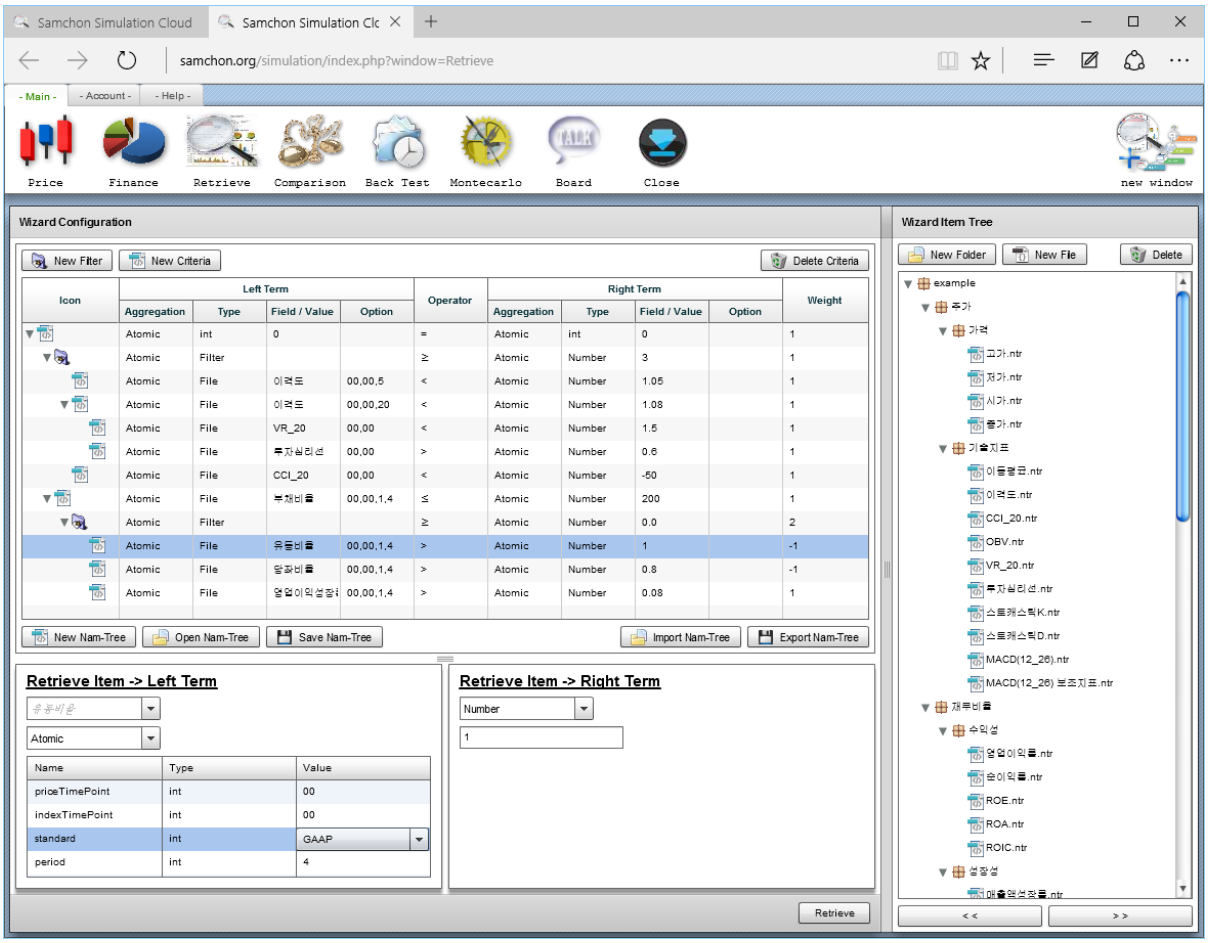
- class NTSide

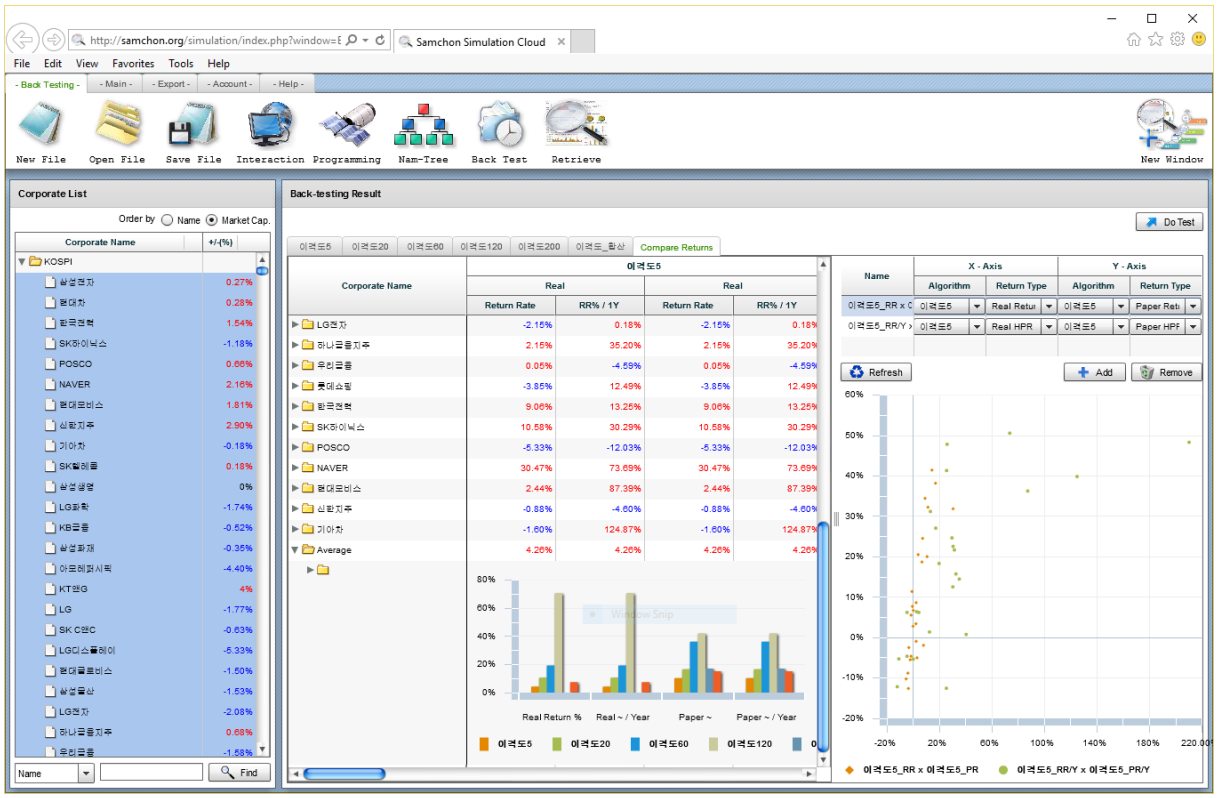
  A side of a conditional expresson.

5.6.1 Detailed Description

Package of Nam-Tree.

Nam-Tree is a module of tree-strucrued complicate logical condition model for realizing A.N.N. (Artificial Neural Network). You can make the ANN model with weight and bias and express not only logical condition by your hand, but also construct the logical condition by automaticaly with optimization of genetic algorithm and grid method.





The basic pricinple of Nam-Tree are follow:

1. A conditional expression with weight

   F(x) = (NTSide < NTSide ? 1: 0) X weight

   NTCrietria is made up for conditional expression.

   - When the expression is true, returns 1 ∗ weight
   - When the expression is false, returns 0

2. Hierarchical relationship

   - In vertical relationship: multiply (X)
   - In horizontal relationship: plus (+)

   With that rule, you can make enormous conditions. I can sure there's not any condition can'be be expressed by the model.

3. Making bias

   Just make a NTCriteria returns only true.

   It's the bias returns only weight.

4. Explore

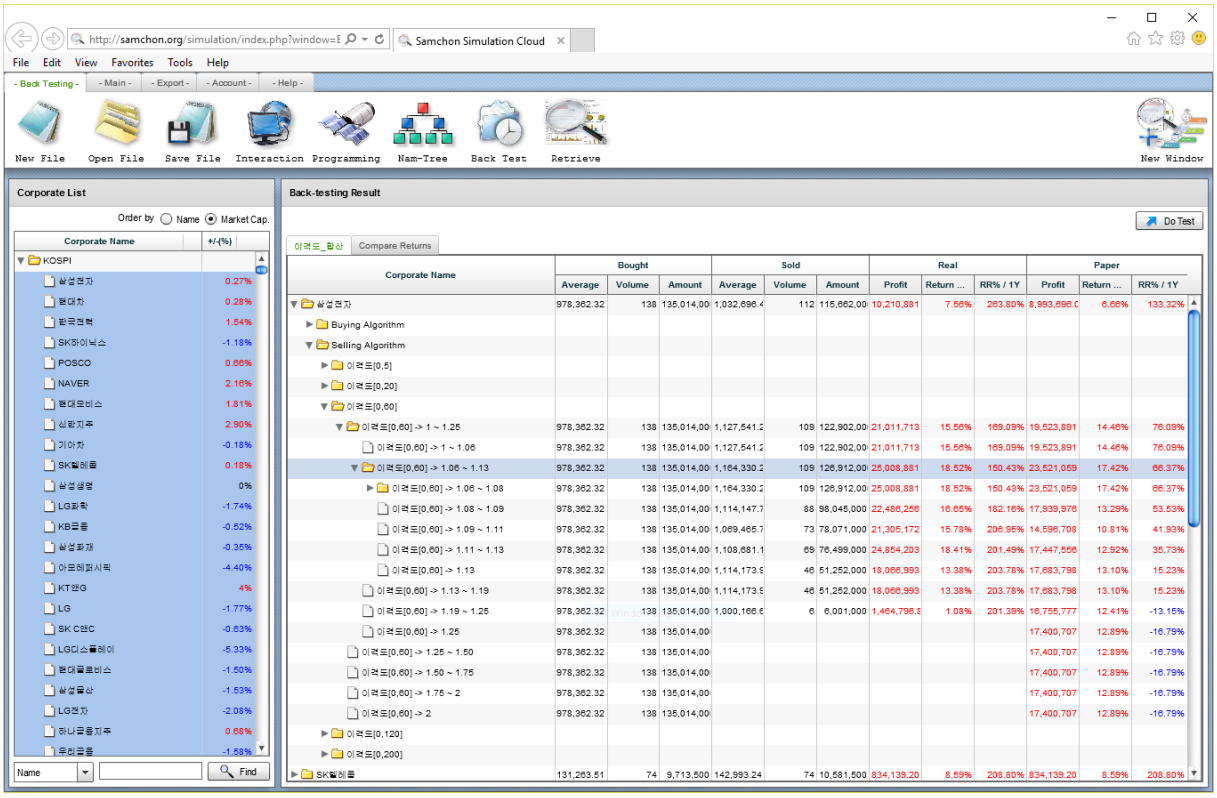
   (a) Exporing in a NTCriteria (optimize a side)

   Make a NTSide to be nullptr, then NTCriteria will explore the best value.

   Nam-Tree will calculate the condtions from minimum to maximum in INTExplore reach to the precision, by the method of multi-demensional grid.

   (b) Exploring parameter in NTSide

   If you set the parameterMNap to be empty, Nam-Tree will explore the best parameter until reach to the precision in INTExploreParameter from minimum to maximum.

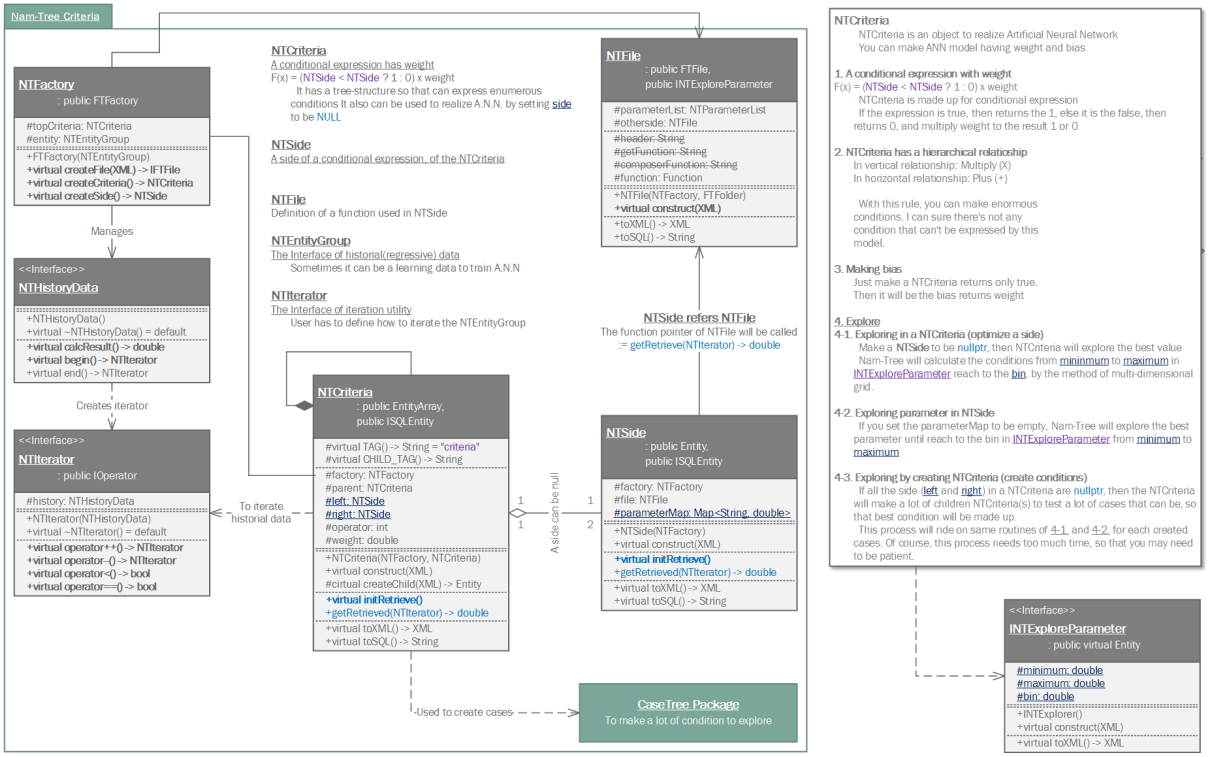


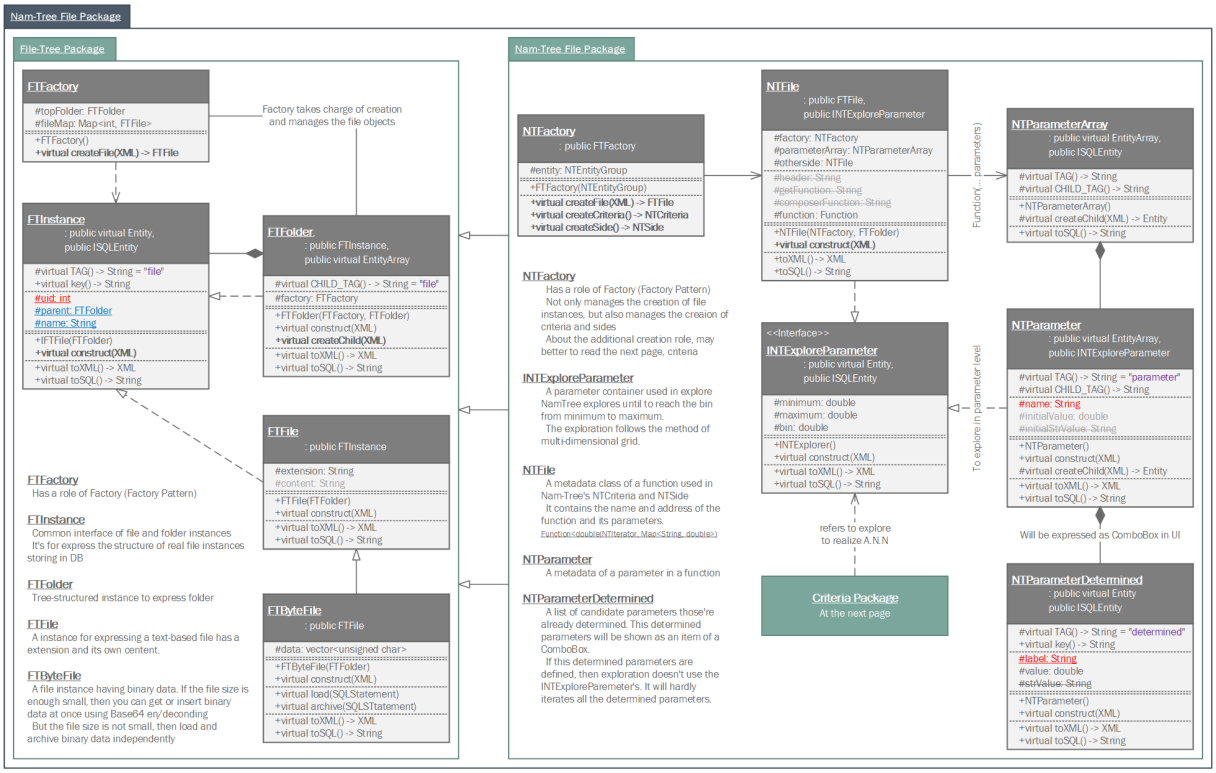
   (c) Exploring by creating NTCriteria

   (Creates lots of hierarchical conditions and tests them)

If all the side (left and right) in a NTCriteria are nullptr, the NTCriteria will make a lot of children NT↩ Criteria(s) to test lots of cases, so that derives the best condition set will be made up.

This process will ride on same routines of 4-1 and 4-2, for each created cases. Of course, this process needs too much time, so that you may need to be patient.





Note

Previous version of the Samchon Framework, NTFile class had a function script on the NTFile and Nam-Tree module compiled the script like Nam-Tree moudle of Flex standalone that is keeping the compiling method.

However, modern Samchon Framework's C++ NTFile class doesn't have a function script and does not compile. I don't know a way to compile a script in lots of operating systems, so I dropped the compiling method. Until update, NTFactory has function pointers in a Dictionary and NTFile reference them. Users must put own methods into the Dictionary by their hands.

I will solve the problem sooon. Until next generation of Samchon Framework, put your own function pointers to a Dictionary in NTFactory by your hand please. Sorry for my unripe skill on programming.

Author

Jeongho Nam

5.7 samchon::protocol Namespace Reference

Package of network protocol and libraries.

Namespaces

- master

  Package for external system, within the framework of master.
- service

  Package of cloud service as a server.
- slave

  Package of external system, within the framework of slave.

Classes

- class Entity

  An entity, a standard data class.
- class EntityArray

  An Entity and a static array containing Entity objects.
- class EntityGroup

  An Entity and a container of children Entity objects.
- class EntityList

  An Entity and a static list containing Entity objects.
- class ExternalClient

  A network driver for an external client.
- class ExternalClientArray

  An array of ExternalClient(s)
- class ExternalServer

  A network driver for an external server.
- class ExternalServerArray

  An array of ExternalServer(s)
- class ExternalSystem

  A network driver for an external system.
- class ExternalSystemArray

An array of ExternalSystem(s).

- class ExternalSystemRole

  A role belongs to an external system.
- class FlashPolicyServer

  A flash policy server.
- class IClient

  An interface for a client.
- class IEntityChain

  A chain of entity.
- class IEntityGroup

  An iternface for entity group.
- class IEntityPtrGroup
- class IHTMLEntity

  An interface supporting conversion to html.
- class Invoke

  Standard message of network I/O.
- class InvokeHistory

  A history of an Invoke message.
- class InvokeHistoryArray

  An array of InvokeHistory.
- class InvokeParameter

  A parameter of an Invoke.
- class IProtocol

  An interface of Invoke message chain.
- class IServer

  An interface of a physical server.
- class ISQLEntity

  An interface supporting DB-I/O.
- class IWebClient

  An interface for a web-client.
- class IWebServer

  An interface for a physical server following web-socket.
- class PRInvokeHistory

  A reported history of an Invoke message.
- class ServerConnector

  A server connector for a physical client.
- class SystemRole

  A role belongs to a system.
- class WebServerConnector

  A web-socket server connector.

Typedefs

- template<typename _Ty = Entity>
  using SharedEntityArray = EntityGroup< std::vector< std::shared_ptr< _Ty >>, _Ty, std::shared_ptr< _Ty > >

  An EntityGroup with vector container and children capsuled in shared pointers.
- template<typename _Ty = Entity>
  using SharedEntityList = EntityGroup< std::list< std::shared_ptr< _Ty >>, _Ty, std::shared_ptr< _Ty > >

  An EntityGroup with list container and children capsuled in shared pointers.

- template<typename _Ty = Entity>
  using UniqueEntityArray = EntityGroup< std::vector< std::unique_ptr< _Ty >>, _Ty, std::unique_ptr< _Ty > >

  An EntityGroup with vector container and children capsuled in unique pointers.

- template<typename _Ty = Entity>
  using UniqueEntityList = EntityGroup< std::list< std::unique_ptr< _Ty >>, _Ty, std::unique_ptr< _Ty > >

  An EntityGroup with list container and children capsuled in unique pointers.

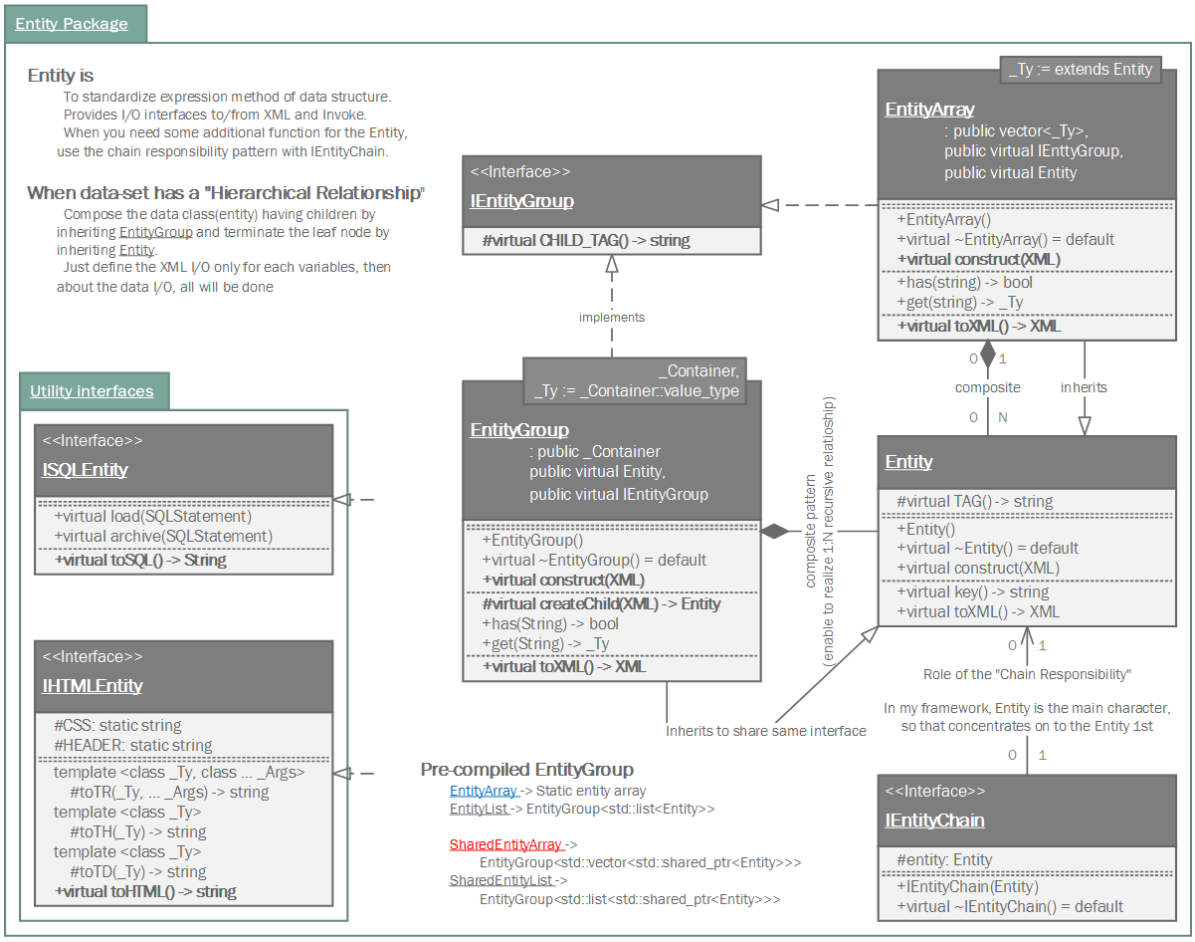
5.7.1 Detailed Description

Package of network protocol and libraries.

- **Entity - data classes with utility methods**

  In Samchon Framework, with entity, boundary and control, entity is the main. The entity package provides I/O interface to/from XML, Invoke and DBMS. If you need some additional function (role of boundary or control) for an entity, use chain of responsibility pattern starting from the entity.

  When data-set has a "hierarchical relationship", do not have the children objects by having a container member variable. Directly inherits container set. Composes the data class (entity) having children by inheriting EntityGroup and terminate the leaf node by inheriting Entity (single object).



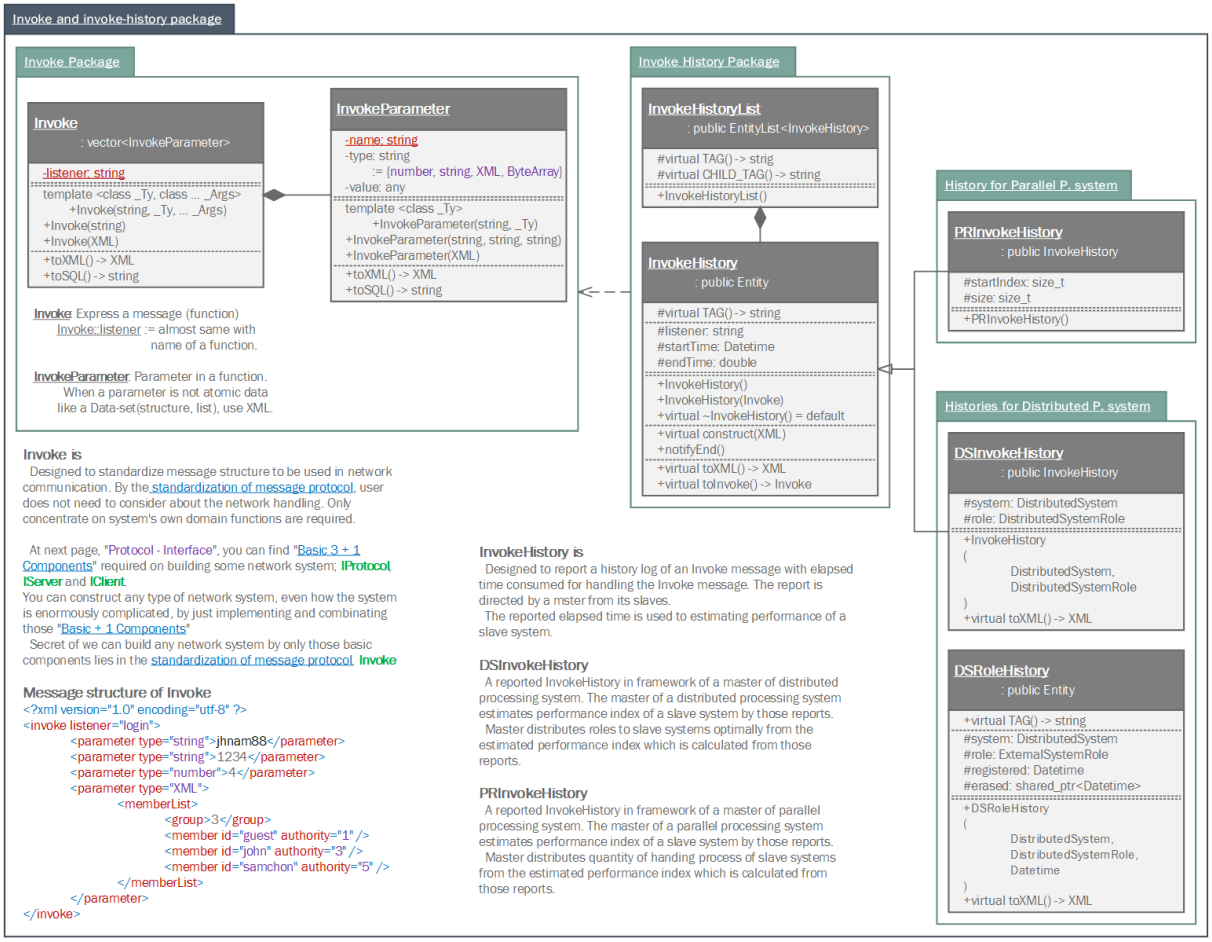
- **Invoke - standard network I/O message**

  Invoke is a standard message structure using network I/O in Samchon Framework.

The Invoke message has an XML structure like the piacture in below. We can enjoy lots of benefits by the normalized and standardized message structure used in network I/O.

The greatest advantage is that we can make any type of network system, even how the system is enourmously complicated. As network communication message is standardized, we only need to concentrate on logical relationships between network systems. We can handle each network system like a object (class) in OOD. And those relationships can be easily designed by using design pattern.

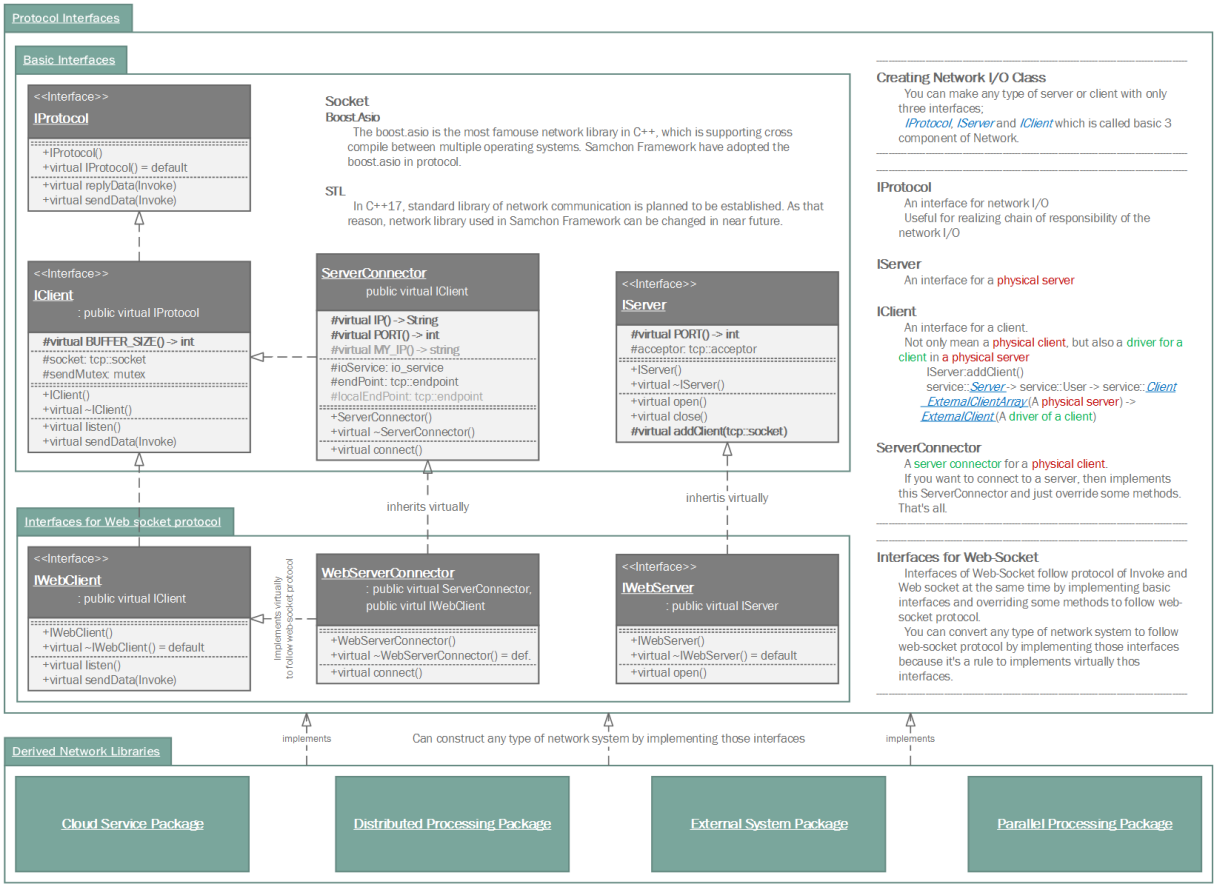
Furthermore, The module invoke not only provides parsing and converting from/to Invoke message, but also history classes can be used to archiving log or estimating performance of a system. The purpose of estimating performance a system, especially, used in module of distributed processing and parallel processing systems.

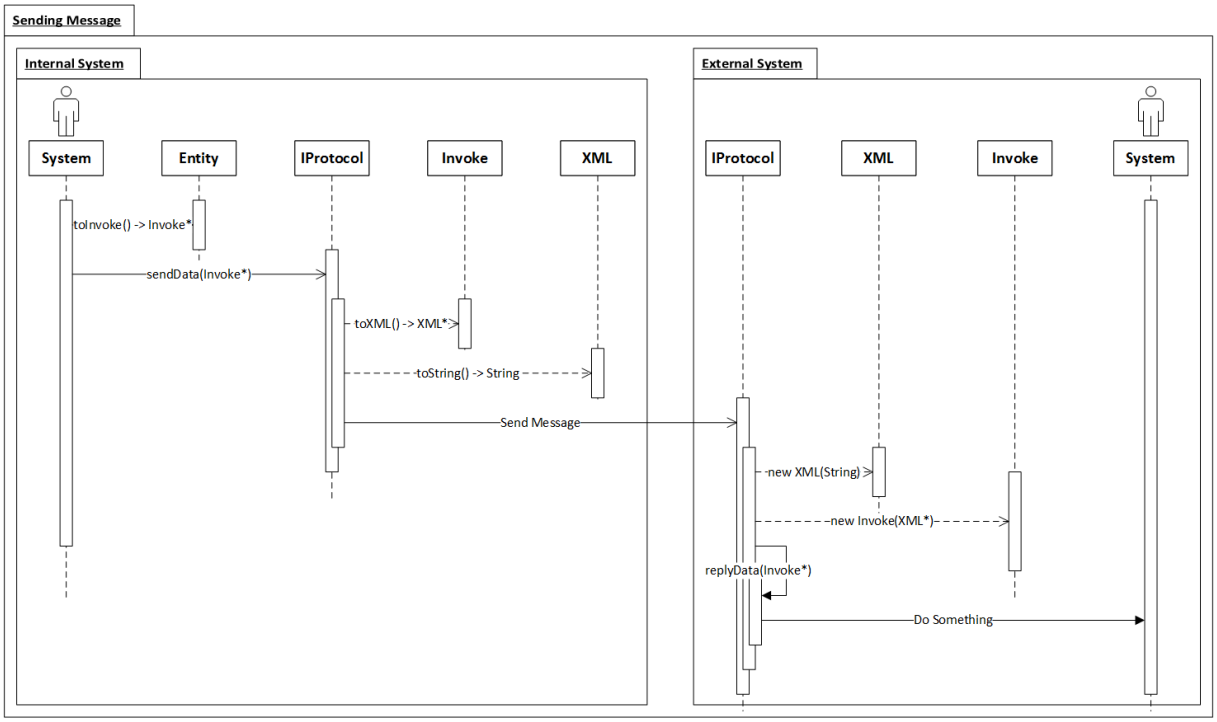


- **Intefaces**

You can make any type of network system with only three + one interfaces; IProtocol, IServer and IClient + ServerConnector which are called basic 3 + 1 components. As I've mentiond in previous invoke module, because of standardization of message of network I/O, we only need to concentrate on logical relationship like handling OOD objects

The basic 3 + 1 components are top level abstract interfaces for designing network system like handling OOD objects. When you see the protocol module from a different view, out of framework, you can find that all the other classes in protocol are another examples combinationing and utilizing those basic 3 + 1 components (interaces).





- **External System**

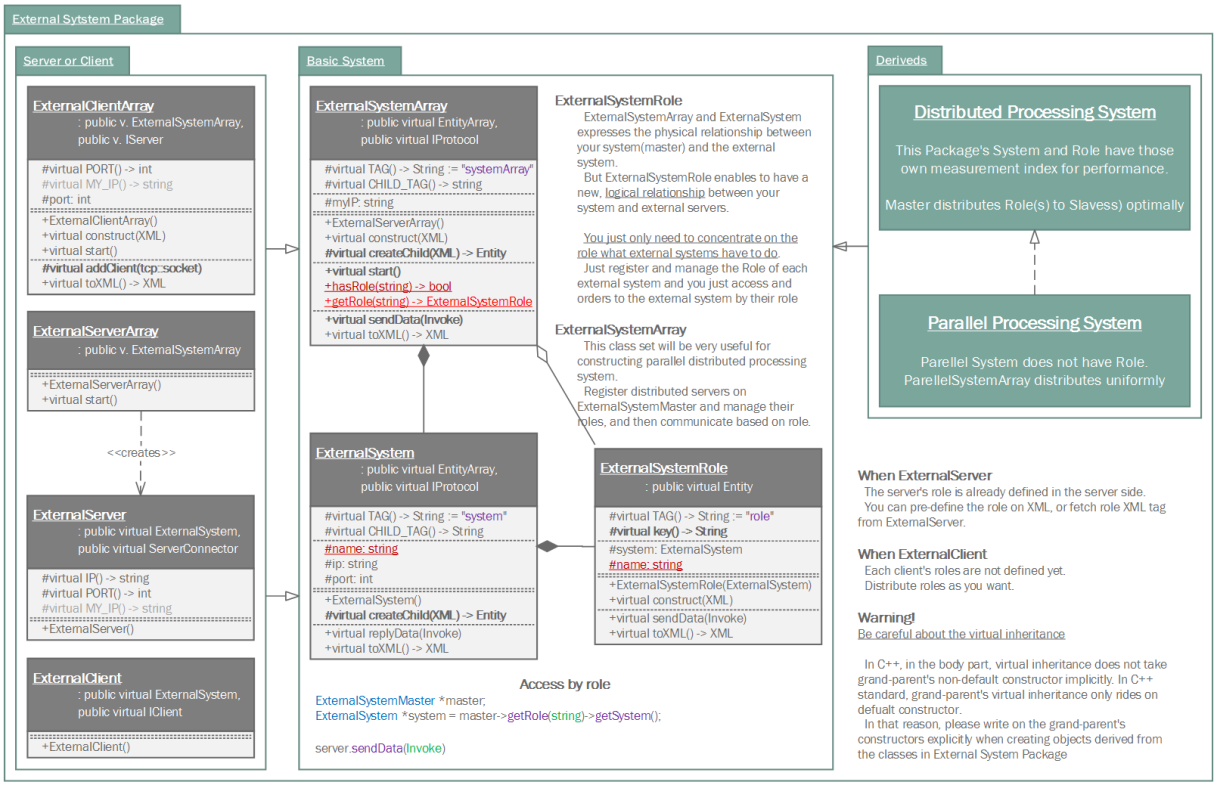
  Module *external_system* provides interfaces for interaction with external network system. Although, the module *external_system* acts boundary as main role, what you've to concentrate on is the entity. Samchon Framework takes responsibility of network communication and you only consider about relationship and role of each *external network systems*.

ExternalSystem objects are managed by ExternalSystemArray and the ExternalSystemArray can access to an ExternalSystemRole belongs to an ExternalSystem directly. When you send an Invoke message to ExternalSystemArray, the ExternalSystemArray finds matched ExternalSystemRole and the External↩ SystemRole shifts the network I/O responsibility to belonged ExternalSystem.

The relationship called as "Proxy Pattern". With the pattern, "Proxy", you can concentrate on roles irrespective of where each role is belonged to (you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem).

- ExternalSystemArray::sendData() -> ExternalSystemRole(Proxy)::sendData() -> ExternalSystem↩ ::sendData()
- ExternalSystem::replyData() -> ExternalSystemRole(Proxy)::replyData()

Whether using the "Proxy pattern" is on your mind in *external_system module* level. "Proxy pattern" is recommend to use in *external_system* module, but not forced. However, since *parallel_processing_system* module, you've to follow the pattern.

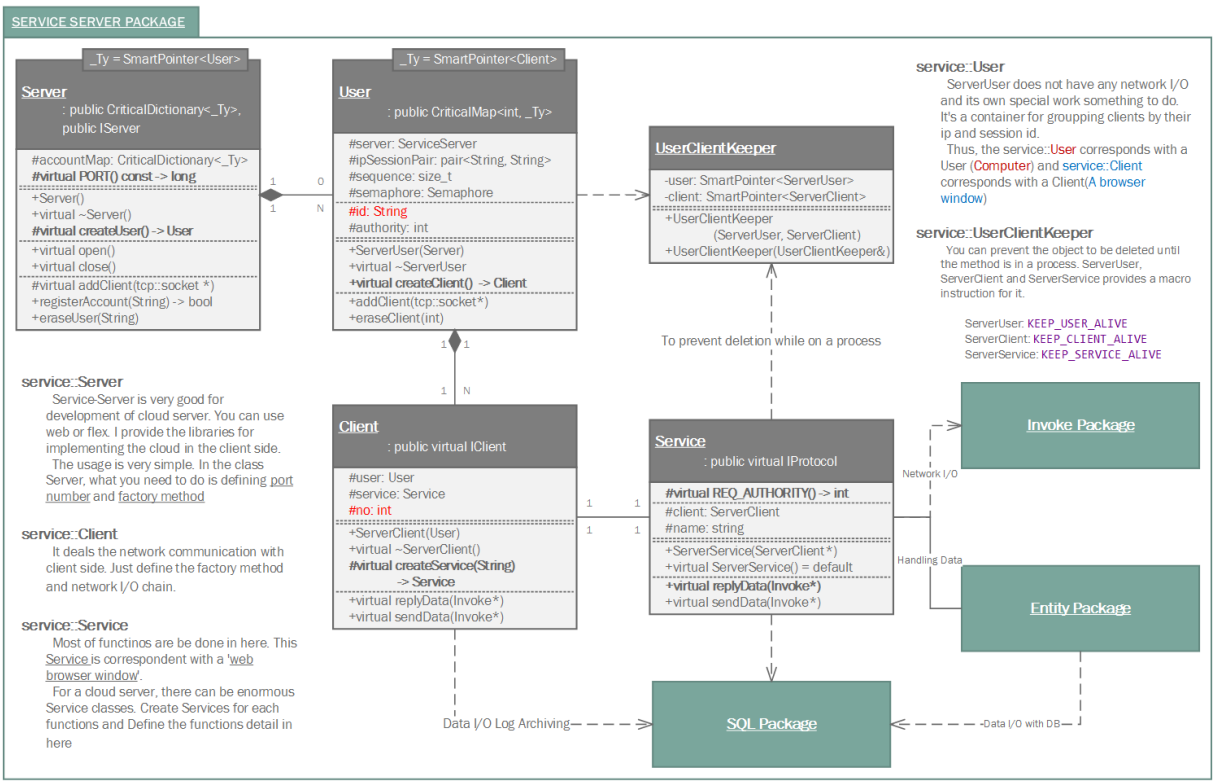


- **Packages in protocol**

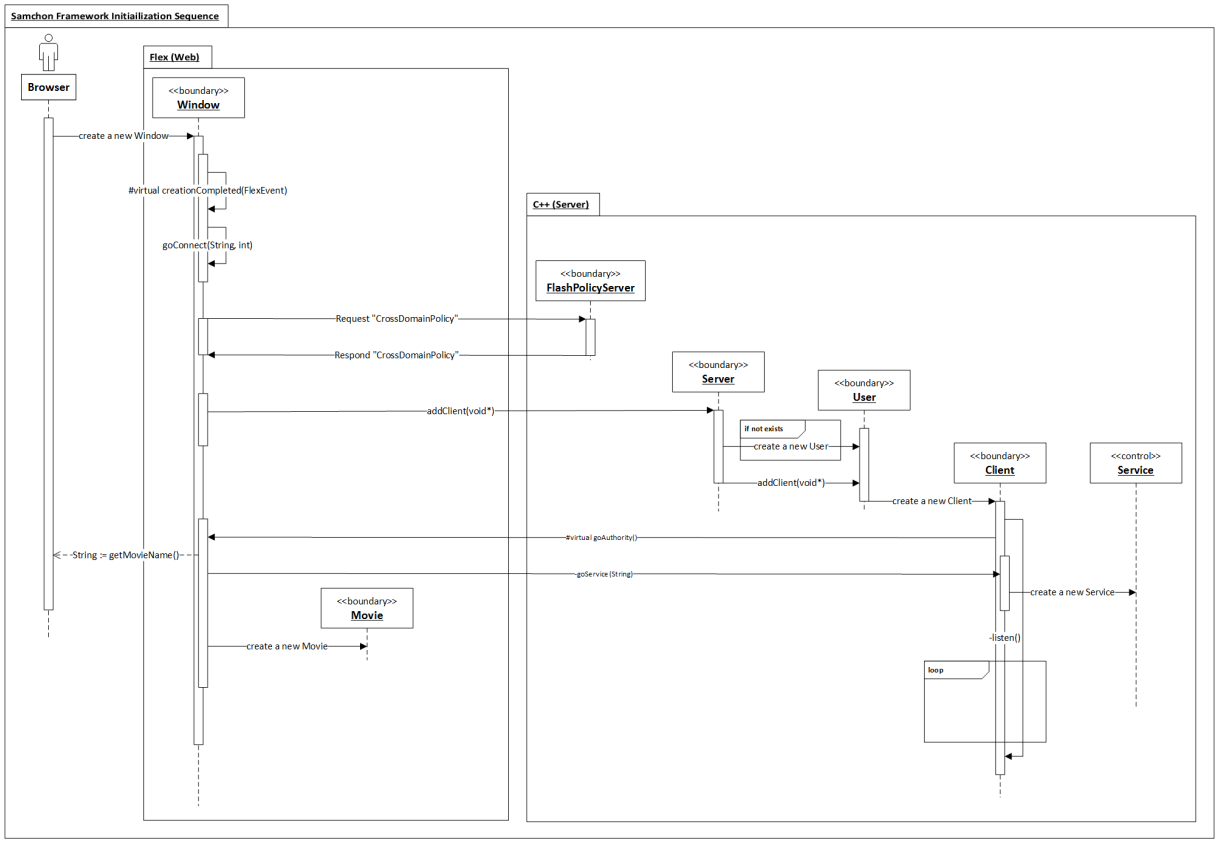
Cloud service

The "service" package is for realizing cloud service. In Samchon Framework, realizing cloud service, there're two classes representing physical server and client as service::Server and service::Client. Between the two classes representing boundaries, there is a class representing logical conception user as service::User. In the terminal node under the service::Client, service::Service class represents a control. The default protocol of the service package is not 'web socket'. The default only has message protocol of Invoke, does not have handshake or another pre-defined protocol. By the default protocol, you can connect to the cloud server (built from c++, by service package) as a client by a program or plug-in like Flex or C#. However you can't connect by web-browser only allowing socket protocol as web-socket. To build a cloud service to follow web-socket protocol, inherits derviced classes not from service::Server, service::Client, but from service::WebServer, service::WebClient or implements IWebServer, IWebClient which are derived from service::Server and service::Client. **Class Diagram**

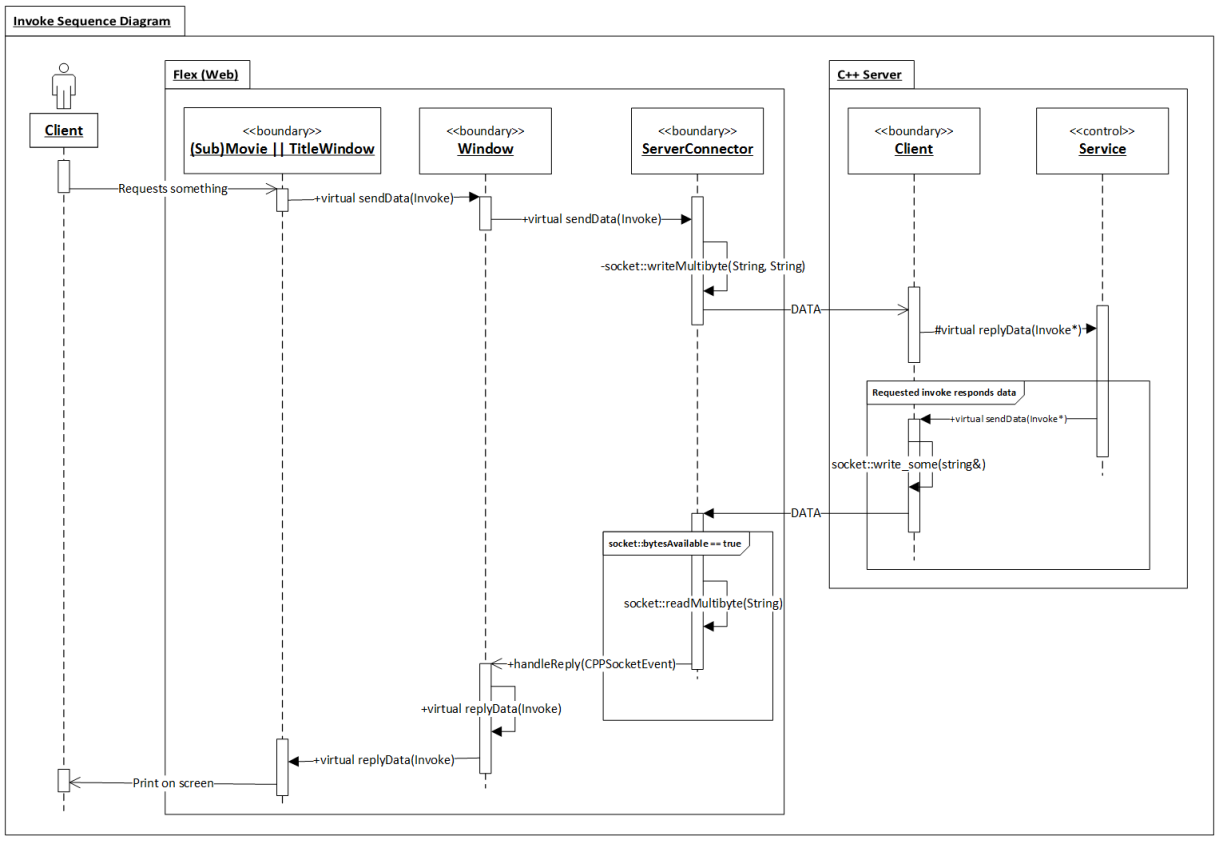
–



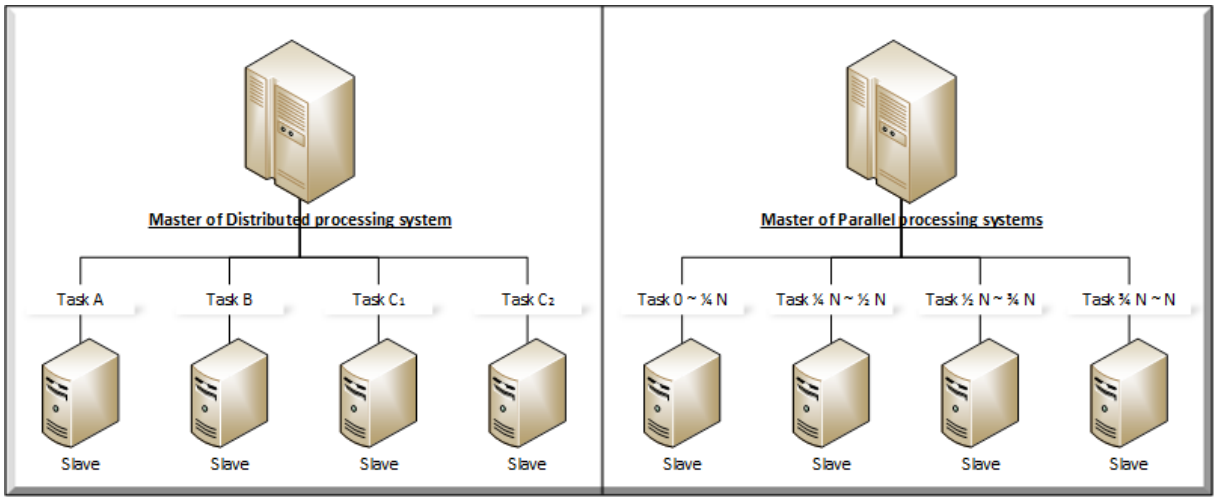
Sequence Diagram - A client's connection



Sequence Diagram - Network communication



– Master system



In master package, provides distributed and parallel processing system modules which are in framework of master. With classes in master package, you can realize any type of distributed or parallel processing system in master side, even how enormouse scale those have.

Distributed Processing System

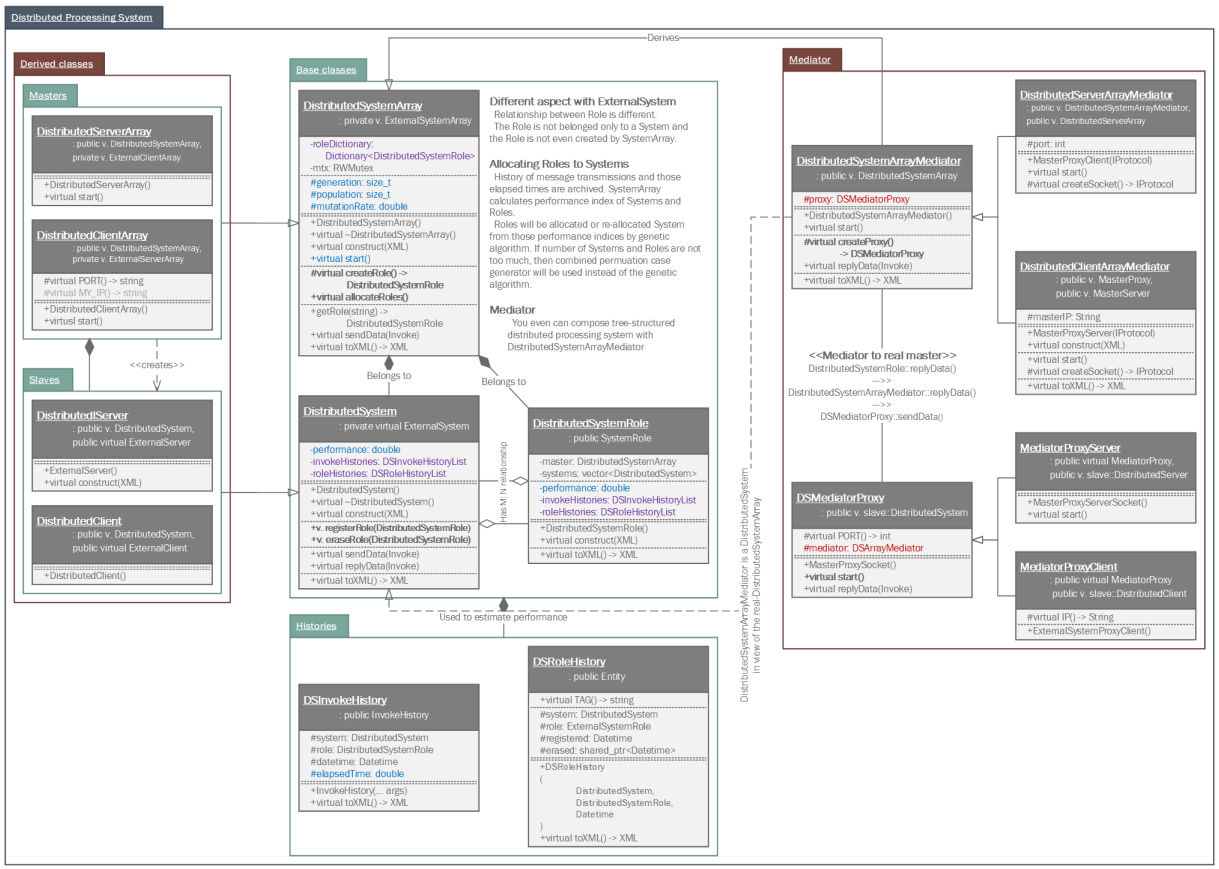
A distributed processing system of master, a master (master::DistributedSystemArray) allocates roles (master::DistributedSystemRole) to systems (master::DistributedSystem). User does not consider about which roles are allocated to which systems. User just accesses to a *role* and send or reply message from those roles.

* It's the proxy pattern have mentioned in external system
  ExternalSystem objects are managed by ExternalSystemArray and the ExternalSystemArray can access to a role belongs to a system directly. When you send an Invoke message to External↩SystemArray, the ExternalSystemArray finds matched ExternalSystemRole and the External↩SystemRole shifts the network I/O responsibility to belonged ExternalSystem. This relationship called "Proxy Pattern". By the pattern, "Proxy", you can concentrate on roles irrespective of where each role is belonged to.

  · ExternalSystemArray::sendData() -> ExternalSystemRole(Proxy)::sendData() -> External↩System::sendData()

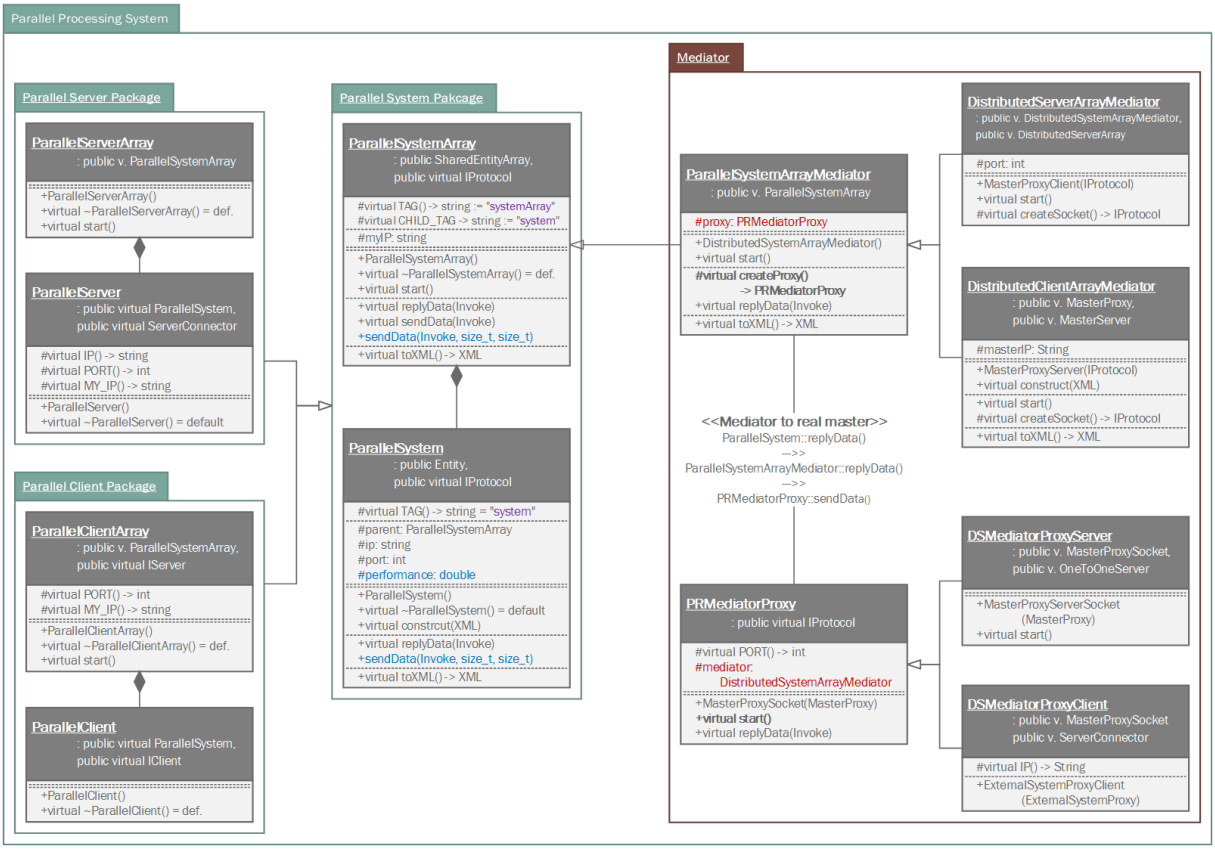
  · ExternalSystem::replyData() -> ExternalSystemRole(Proxy)::replyData()

However, unlike the case of managing roles from ExternalSystemArray, in DistributedSystemArray, roles can be allocated in multiple systems duplicately. And the roles can be moved to another systems. Those management and allocation of roles are determined by estimation of performance of each system, and required performance of each role.



Parallel Processing System

Unlike distributed processing system, the parallel processing system is not complicate. It has very simple logic. About requested processes, master(master::ParallelSystemArray) allocates each process to each slave system by their own performance index have estimated.



– **Slave system**

A package for composing distributed or parallel processing system in slave side.

Slave system get orders from master, processes the orders and report the result with its elapsed time for estimating performance of the slave system.

cpp/protocol_slave.png

Note

- Boost Libraries: http://www.boost.org/
- Boot.Asio: http://www.boost.org/doc/libs/1_59_0/doc/html/boost_asio.html

Author

Jeongho Nam

5.7.2 Typedef Documentation

5.7.2.1 SharedEntityArray

An EntityGroup with vector container and children capsuled in shared pointers.

Template Parameters

_Ty	A type of children Entity. Must be a class derived from an Entity or Entity itself.

For a case of inheritance of SharedEntityArray and the inherited SharedEntityArray also contains entity objects which are inherited from original child entity type, SharedEntityArray supports macros changing returned type of accessor methods.

- SHARED_ENTITY_ARRAY_ELEMENT_ACCESSOR_HEADER
- SHARED_ENTITY_ARRAY_ELEMENT_ACCESSOR_BODY

See also

samchon::protocol

Author

Jeongho Nam

5.7.2.2 using SharedEntityList = EntityGroup < std::list<std::shared_ptr<_Ty>>, _Ty, std::shared_ptr<_Ty> >

An EntityGroup with list container and children capsuled in shared pointers.

Template Parameters

_Ty	A type of children Entity. Must be a class derived from an Entity or Entity itself.

See also

samchon::protocol

Author

Jeongho Nam

5.7.2.3 using UniqueEntityArray = EntityGroup < std::vector<std::unique_ptr<_Ty>>, _Ty, std::unique_ptr<_Ty> >

An EntityGroup with vector container and children capsuled in unique pointers.

Template Parameters

_Ty	A type of children Entity. Must be a class derived from an Entity or Entity itself.

For a case of inheritance of UniqueEntityArray and the inherited UniqueEntityArray also contains entity objects which are inherited from original child entity type, UniqueEntityArray supports macros changing returned type of accessor methods.

- UNIQUE_ENTITY_ARRAY_ELEMENT_ACCESSOR_HEADER
- UNIQUE_ENTITY_ARRAY_ELEMENT_ACCESSOR_BODY

Note

std::unique_ptr doesn't allow copy construction. It allows only move construction. When inserts children objcets not by construct() method but by your hand, Be careful to insert deriving copy construction. You've use move constructor.

- *UniqueEntityArray<?>::push_back(move(std::unique_ptr<?>(new ?(...))));*

See also

samchon::protocol

Author

Jeongho Nam

5.7.2.4 using UniqueEntityList = EntityGroup < std::list<std::unique_ptr<_Ty>>, _Ty, std::unique_ptr<_Ty> >

An EntityGroup with list container and children capsuled in unique pointers.

Template Parameters

_Ty	A type of children Entity. Must be a class derived from an Entity or Entity itself.

Note

std::unique_ptr doesn't allow copy construction. It allows only move construction. When inserts children objcets not by construct() method but by your hand, Be careful to insert deriving copy construction. You've use move constructor.

- *UniqueEntityList<?>::push_back(move(std::unique_ptr<?>(new ?(...))));*

See also

samchon::protocol

Author

Jeongho Nam

5.8 samchon::protocol::master Namespace Reference

Package for external system, within the framework of master.

Classes

- class DistributedClient

  A network driver for a distributed client.
- class DistributedClientArray

  An array of DistributedClient(s).
- class DistributedClientArrayMediator

  A mediator of distributed system between master and slaves.
- class DistributedClientServerMediator

  A distributed slave client for mediation.
- class DistributedServer

  A network driver for a distributed server.
- class DistributedServerArray

  An array of DistributedServer(s).
- class DistributedServerArrayMediator

  A mediator of distributed system between master and slaves.
- class DistributedSlaveServerMediator

  A distributed slave server for mediation.
- class DistributedSlaveSystemMediator

  A distributed slave system for mediation.
- class DistributedSystem

  A network driver for a distributed system.
- class DistributedSystemArray

  An array of distributed system drivers.
- class DistributedSystemArrayMediator

  A mediator of distributed system between master and slaves.

- class DistributedSystemRole

  A role of distributed processing system.
- class DSInvokeHistory

  A reported history of an Invoke message.
- class DSInvokeHistoryList

  A list of DSInvokeHistory.
- class ParallelClient

  A network driver for a parallel client.
- class ParallelClientArray

  An array of ParallelClient(s).
- class ParallelClientArrayMediator

  A mediator of parallel system between master and slaves.
- class ParallelServer

  A network driver for a parallel server.
- class ParallelServerArray

  An array of ParallelServer(s).
- class ParallelServerArrayMediator

  A mediator of parallel system between master and slaves.
- class ParallelSlaveClientMediator

  A parallel slave client for mediation.
- class ParallelSlaveServerMediator

  A parallel slave server for mediation.
- class ParallelSlaveSystemMediator

  A parallel slave system for mediation.
- class ParallelSystem

  A network driver for a parallel system.
- class ParallelSystemArray

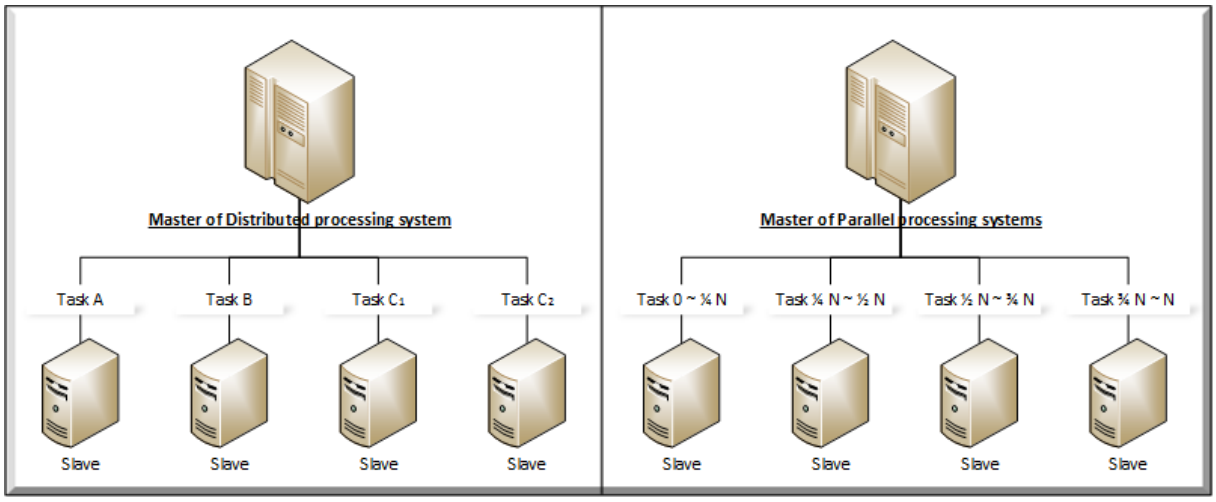
  An array of parallel system drivers.
- class ParallelSystemArrayMediator

  A mediator of parallel system between master and slaves.
- class PRInvokeHistoryArray

  An array of PRInvokeHistory.

5.8.1 Detailed Description

Package for external system, within the framework of master.



In master package, provides distributed and parallel processing system modules which are in framework of master. With classes in master package, you can realize any type of distributed or parallel processing system in master side, even how enormouse scale those have.

Distributed Processing System

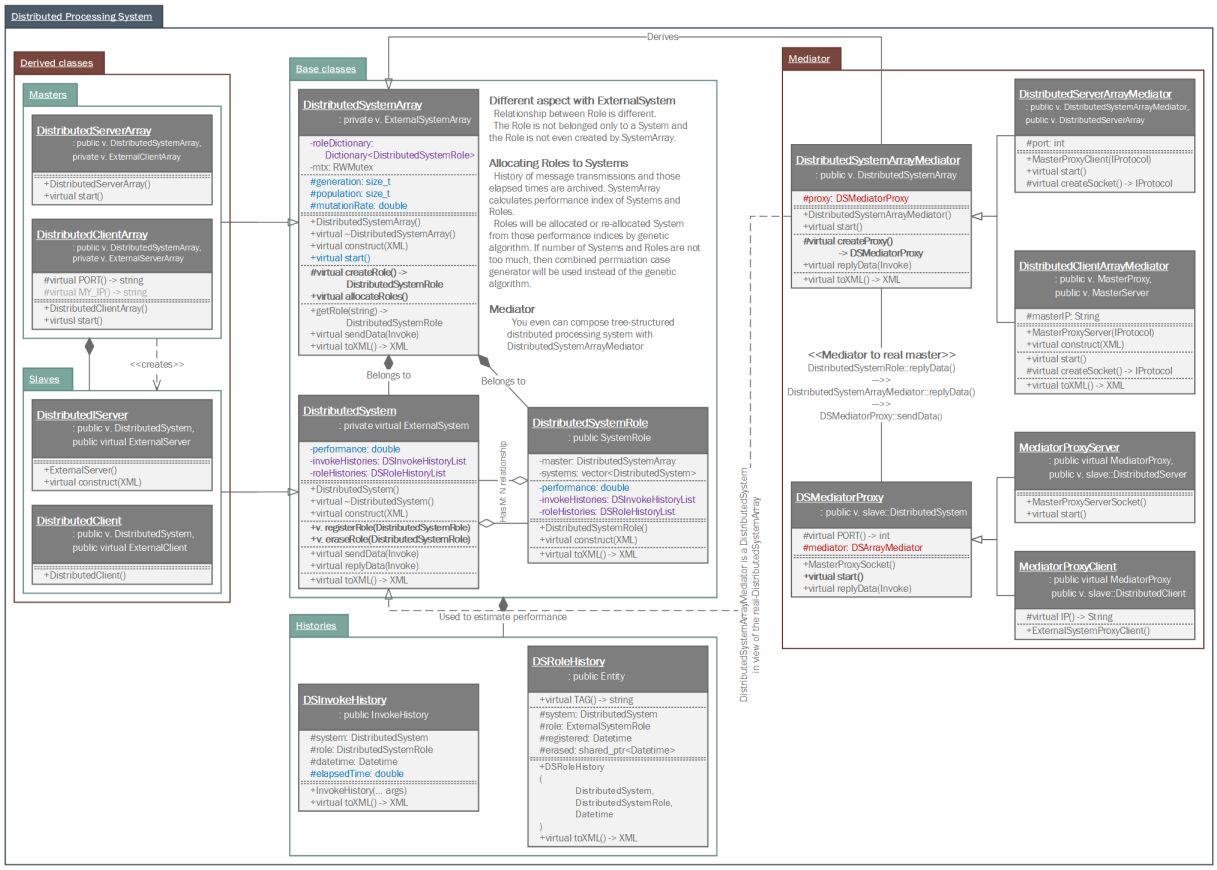
A distributed processing system of master, a master (master::DistributedSystemArray) allocates roles (master↩::DistributedSystemRole) to systems (master::DistributedSystem). User does not consider about which roles are allocated to which systems. User just accesses to a *role* and send or reply message from those roles.

- It's the proxy pattern have mentioned in external system

  ExternalSystem objects are managed by ExternalSystemArray and the ExternalSystemArray can access to a role belongs to a system directly. When you send an Invoke message to ExternalSystemArray, the ExternalSystemArray finds matched ExternalSystemRole and the ExternalSystemRole shifts the network I/O responsibility to belonged ExternalSystem. This relationship called "Proxy Pattern". By the pattern, "Proxy", you can concentrate on roles irrespective of where each role is belonged to.

  - ExternalSystemArray::sendData() -> ExternalSystemRole(Proxy)::sendData() -> ExternalSystem↩::sendData()
  - ExternalSystem::replyData() -> ExternalSystemRole(Proxy)::replyData()

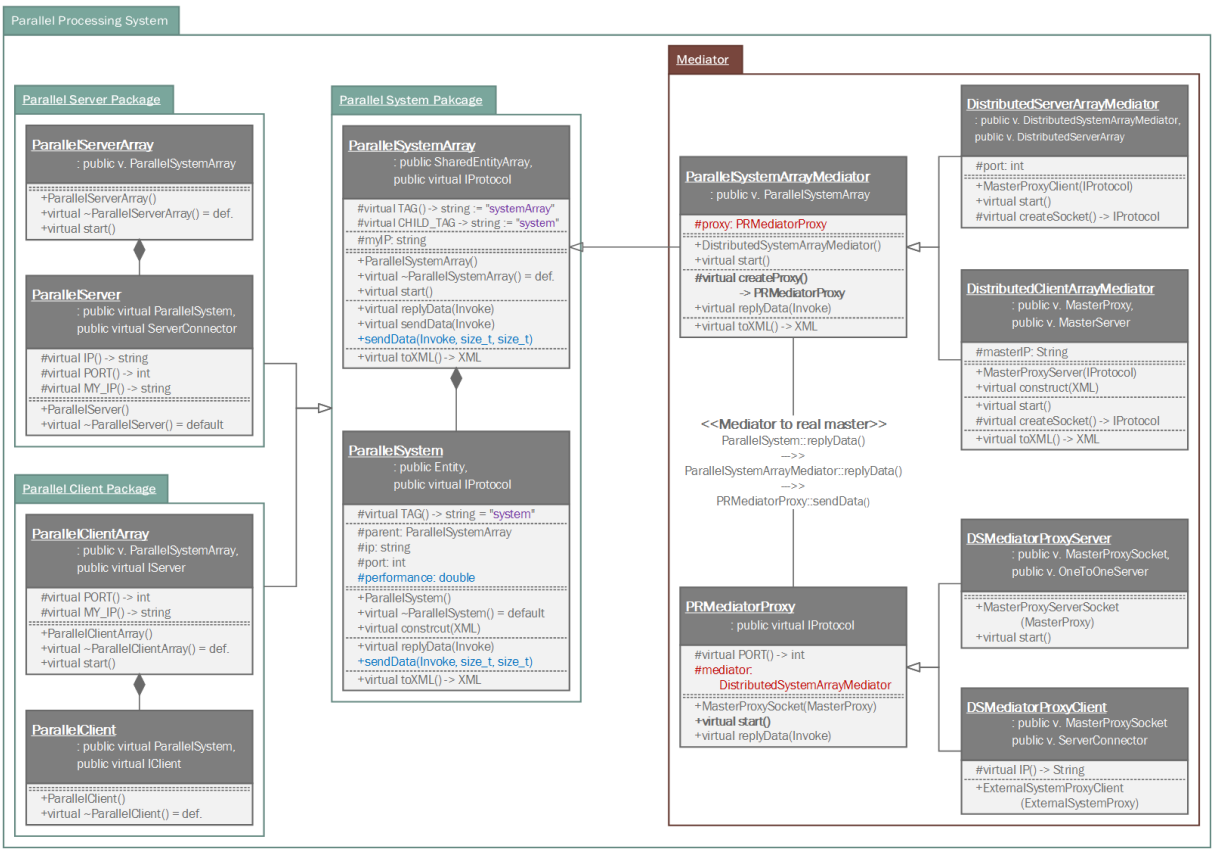
However, unlike the case of managing roles from ExternalSystemArray, in DistributedSystemArray, roles can be allocated in multiple systems duplicately. And the roles can be moved to another systems. Those management and allocation of roles are determined by estimation of performance of each system, and required performance of each role.



Parallel Processing System

Unlike distributed processing system, the parallel processing system is not complicate. It has very simple logic. About requested processes, master(master::ParallelSystemArray) allocates each process to each slave system by

their own performance index have estimated.



Author

Jeongho Nam

5.9 samchon::protocol::service Namespace Reference

Package of cloud service as a server.

Classes

- class Client

  A network boundary with the client in an User.
- class IPUserPair

  Relationship between IP address and User for issuing session.
- class Server

  A server for (cloud) service.
- class Service

  A service.
- class ServiceKeeper

  A keeper blocking destrunction of User and Client.
- class User

  User containing Client(s) with session-id.

5.9.1 Detailed Description

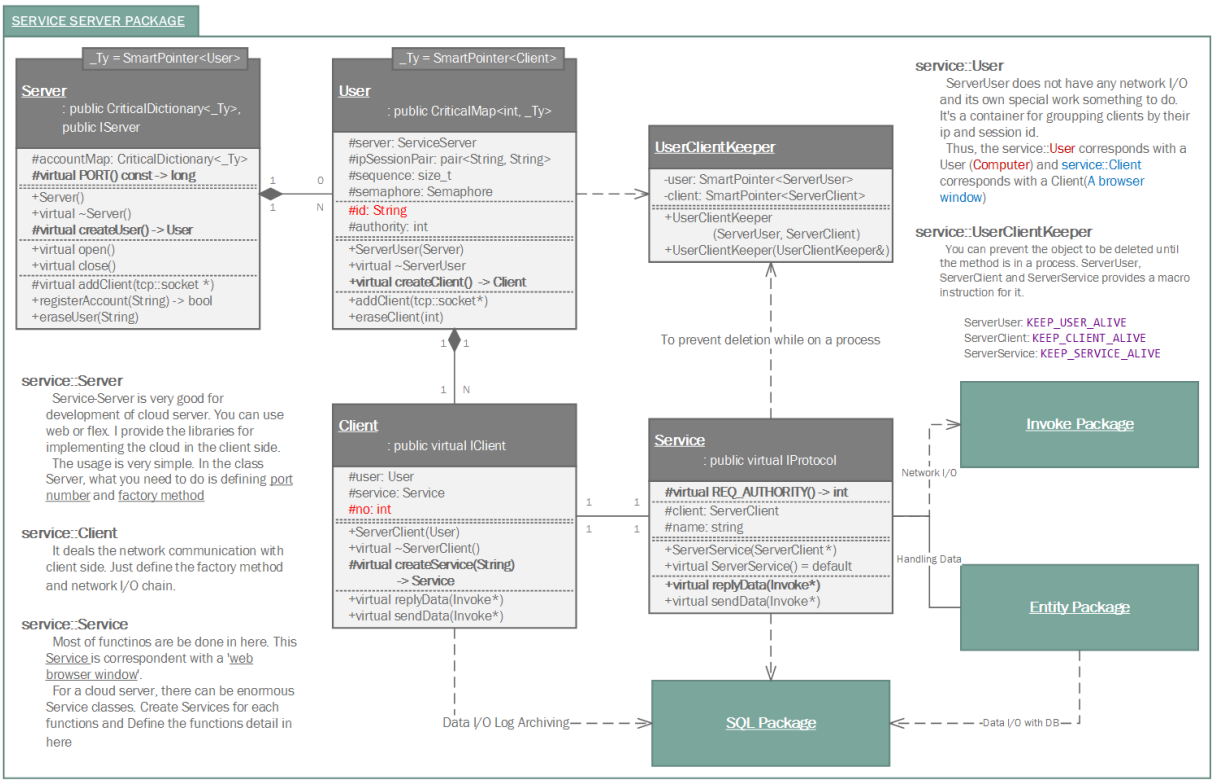
Package of cloud service as a server.

In Samchon Framework, realizing cloud service, there're two classes representing physical server and client as Server and Client. Between the two classes representing boundaries, there is a class representing logical conception user as User. In the terminal node under the Client, Service class represents a control.

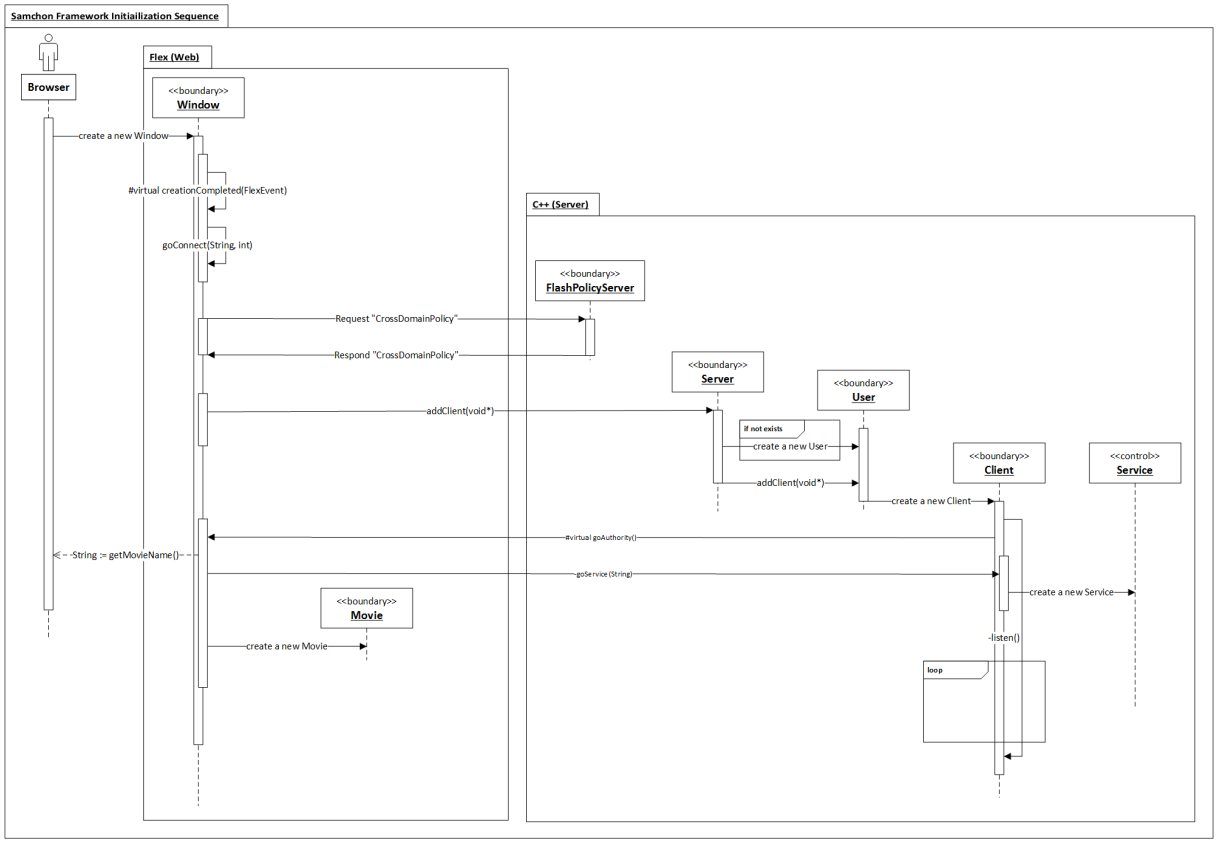
The default protocol of the service package is not 'web socket'. The default only has message protocol of Invoke, does not have handshake or another pre-defined protocol. By the default protocol, you can connect to the cloud server (built from c++, by service package) as a client by a program or plug-in like Flex or C#. However you can't connect by web-browser only allowing socket protocol as web-socket.

To build a cloud service to follow web-socket protocol, inherits derviced classes not from Server, Client, but from WebServer, WebClient or implements IWebServer, IWebClient which are derived from Server and Client.

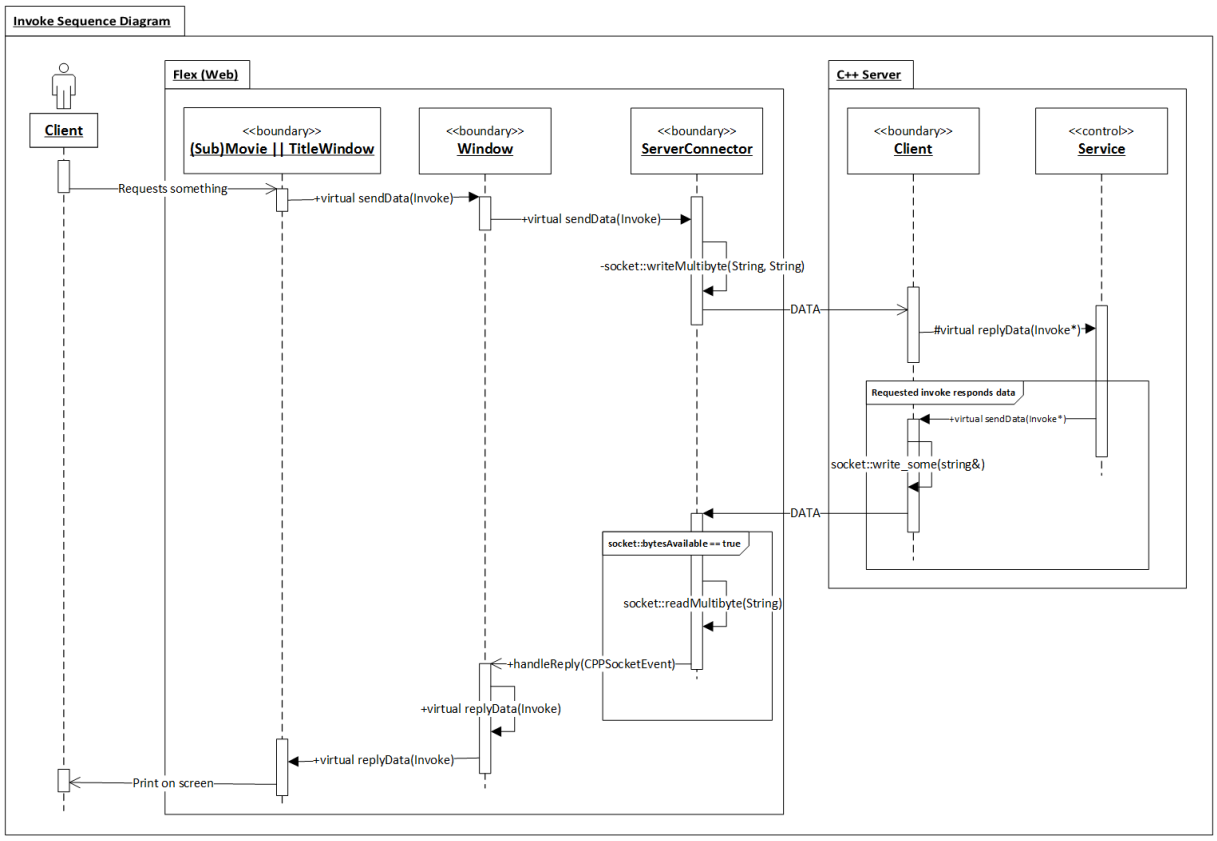
Class Diagram



Sequence Diagram - A client's connection



Sequence Diagram - Network communication



Simple example - Chat service

examples/chat_service/ChatServer.hpp

```
1 #pragma once
2 #include <samchon/protocol/service/Server.hpp>
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace chat_service
9         {
10             class ChatRoomArray;
11             class ChatRoom;
12
22             class ChatServer
23                 : public protocol::service::Server
24             {
25             protected:
26                 typedef protocol::service::Server super;
27
28                 virtual auto PORT() const -> int override
29                 {
30                     return 37749;
31                 };
32
33             private:
37                 ChatRoomArray *roomArray;
38
39             public:
43                 ChatServer();
44                 virtual ~ChatServer();
45
46             protected:
47                 virtual auto createUser() -> protocol::service::User* override;
48
49             public:
53                 auto getChatRoomArray() const -> ChatRoomArray*;
54             };
55         };
56     };
57 };
```

examples/chat_service/ChatUser.hpp

```
1 #pragma once
2 #include <samchon/protocol/service/User.hpp>
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace chat_service
9         {
19             class ChatUser
20                 : public protocol::service::User
21             {
22             public:
26                 ChatUser(protocol::service::Server*);
27                 virtual ~ChatUser() = default;
28
29             protected:
30                 virtual auto createClient() -> protocol::service::Client* override;
31             };
32         };
33     };
34 };
```

examples/chat_service/ChatClient.hpp

```
1 #pragma once
2 #include <samchon/protocol/service/Client.hpp>
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace chat_service
9         {
19             class ChatClient
20                 : public protocol::service::Client
21             {
22             public:
```

```
26                  ChatClient(protocol::service::User*);
27                  virtual ~ChatClient() = default;
28
29              protected:
30                  auto createService(const std::string &) -> protocol::service::Service* override;
31              };
32          };
33      };
34 };
```

examples/chat_service/ChatService.hpp

```
1 #pragma once
2 #include <samchon/protocol/service/Service.hpp>
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace chat_service
9         {
10              class ChatRoom;
11
22              class ChatService
23                  : public protocol::service::Service
24              {
25              private:
26                  typedef protocol::service::Service super;
27
32                  ChatRoom *room;
33
34              public:
38                  ChatService(protocol::service::Client*);
39                  virtual ~ChatService();
40
41                  virtual void replyData(std::shared_ptr<protocol::Invoke>) override;
42              };
43          };
44      };
45 };
```

examples/chat_service/ListService.hpp

```
1 #pragma once
2 #include <samchon/protocol/service/Service.hpp>
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace chat_service
9         {
10              class ListService;
11
21              class ListService
22                  : public protocol::service::Service
23              {
24              private:
25                  typedef protocol::service::Service super;
26
27              public:
31                  ListService(protocol::service::Client*);
32                  virtual ~ListService() = default;
33
34                  virtual void replyData(std::shared_ptr<protocol::Invoke> invoke) override;
35
36              private:
42                  void createRoom(const std::string &);
43
44                  void handleRoomArray(std::shared_ptr<library::XML>);
45                  void handleMakeRoom(bool);
46                  void handleRoomArray(bool);
47              };
48          };
49      };
50 };
```

examples/chat_service/ChatRoomArray.hpp

```
1 #pragma once
2 #include <samchon/library/CriticalDictionary.hpp>
3 #include <samchon/protocol/IEntityGroup.hpp>
4
```

```
namespace samchon
{
    namespace library { class XML; };
    namespace protocol { class Invoke; };

     namespace example
     {
         namespace chat_service
         {
             class ChatRoom;

             class ChatServer;
             class ChatUser;

             class ChatRoomArray
                 : public CriticalDictionary<std::shared_ptr<ChatRoom>>,
                 public protocol::IEntityGroup
             {
             protected:
                 typedef CriticalDictionary<std::shared_ptr<ChatRoom>> super;

                 virtual auto TAG() const -> std::string { return "roomArray"; };
                 virtual auto CHILD_TAG() const -> std::string { return "room"; };

             private:
                 ChatServer *server;

             public:
                 /* ----------------------------------------------------------------
                     CONSTRUCTORS
                 ---------------------------------------------------------------- */
                 ChatRoomArray(ChatServer*);
                 virtual ~ChatRoomArray() = default;

                 /* ----------------------------------------------------------------
                     NOTIFIER
                 ---------------------------------------------------------------- */
                 void notify();

             private:
                 /* ----------------------------------------------------------------
                     EXPORTERS
                 ---------------------------------------------------------------- */
                 auto toXML() const -> std::shared_ptr<library::XML>;

                 auto toInvoke() const -> std::shared_ptr<protocol::Invoke>;
             };
         };
     };
};
```

examples/chat_service/ChatRoom.hpp

```
#pragma once
#include <samchon/protocol/Entity.hpp>
#include <samchon/protocol/IProtocol.hpp>

#include <string>
#include <samchon/library/CriticalSet.hpp>


namespace samchon
{
    namespace example
    {
        namespace chat_service
        {
            class ChatUser;
            class ChatService;

            class ChatRoomArray;

            class ChatRoom
                : public protocol::Entity,
                public protocol::IProtocol
            {
            protected:
                typedef protocol::Entity super;

                virtual auto TAG() const -> std::string { return "room"; };

            private:
                ChatRoomArray *roomArray;

                std::string name;

```

```
57                  ChatUser *host;
58
62                  library::CriticalSet<ChatService*> participants;
63
64              public:
65                  /* ----------------------------------------------------------
66                      CONSTRUCTORS
67                  ---------------------------------------------------------- */
75                  ChatRoom(ChatRoomArray*, const std::string &, ChatUser*);
76                  virtual ~ChatRoom() = default;
77
82                  void registerClient(ChatService*);
83
88                  void eraseClient(ChatService*);
89
90                  /* ----------------------------------------------------------
91                      CHAIN OF RESPONSIBILITY
92                  ---------------------------------------------------------- */
93                  virtual void replyData(std::shared_ptr<protocol::Invoke>) override;
94                  virtual void sendData(std::shared_ptr<protocol::Invoke>) override;
95
96              public:
97                  /* ----------------------------------------------------------
98                      GETTERS
99                  ---------------------------------------------------------- */
100                  virtual auto toXML() const -> std::shared_ptr<library::XML> override;
101              };
102          };
103      };
104 };
```

examples/chat_service/ChatMessage.hpp

```
1 #pragma once
2 #include <set>
3 #include <string>
4
5 #include <samchon/protocol/Entity.hpp>
6
7 namespace samchon
8 {
9     namespace example
10     {
11         namespace chat_service
12         {
24             class ChatMessage
25                 : public protocol::Entity
26             {
27             protected:
28                 typedef protocol::Entity super;
29
30                 virtual auto TAG() const -> std::string
31                 {
32                     return "message";
33                 };
34
35             private:
39                 std::string orator;
40
45                 std::string listener;
46
50                 std::string message;
51
52             public:
53                 /* ----------------------------------------------------------
54                     CONSTRUCTORS
55                 ---------------------------------------------------------- */
59                 ChatMessage();
60                 virtual ~ChatMessage() = default;
61
62                 virtual void construct(std::shared_ptr<library::XML>);
63
64                 /* ----------------------------------------------------------
65                     GETTERS
66                 ---------------------------------------------------------- */
71                 auto getListener() const -> std::string;
72
73                 virtual auto toXML() const -> std::shared_ptr<library::XML>;
74             };
75         };
76     };
77 };
```

examples/chat_service/ChatServer.cpp

examples/chat_service/ChatUser.cpp

examples/chat_service/ChatClient.cpp

examples/chat_service/ChatService.cpp

```
#include "ChatService.hpp"
#include "ChatRoom.hpp"

#include <samchon/protocol/Invoke.hpp>

using namespace std;
using namespace samchon::protocol;
using namespace samchon::protocol::service;
using namespace samchon::example::chat_service;

ChatService::ChatService(Client *client)
    : super(client)
{

}
ChatService::~ChatService()
{
    if(room == nullptr)
        return;

    room->eraseClient(this);
}

void ChatService::replyData(shared_ptr<Invoke> invoke)
{
    if(invoke->getListener() == "sendMessage")
        room->replyData(invoke);
}
```

examples/chat_service/ListService.cpp

examples/chat_service/ChatRoomArray.cpp

```
#include "ChatRoomArray.hpp"
#include "ChatRoom.hpp"

#include "ChatServer.hpp"
#include "ChatUser.hpp"
#include "ChatClient.hpp"
#include "ListService.hpp"

#include <thread>
#include <samchon/library/XML.hpp>
#include <samchon/protocol/Invoke.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;
using namespace samchon::example::chat_service;

ChatRoomArray::ChatRoomArray(ChatServer *server)
    : super(),
    IEntityGroup()
{
    this->server = server;
}

void ChatRoomArray::notify()
{
    shared_ptr<Invoke> &invoke = this->toInvoke();

    UniqueReadLock uk(get_allocator().getMutex());
    for (auto it = server->begin(); it != server->end(); it++)
    {
        auto user = it->second;
```

```

        for (auto u_it = user->begin(); u_it != user->end(); u_it++)
        {
            auto client = u_it->second;
            auto service = client->getService();

            if (dynamic_cast<ListService>(service) != nullptr)
                thread(&IProtocol::sendData, service, invoke).detach();
        }
    }
}

auto ChatRoomArray::toXML() const -> shared_ptr<XML>
{
    shared_ptr<XML> xml(new XML());
    xml->setTag(TAG());

    UniqueReadLock uk(get_allocator().getMutex());
    for(auto it = begin(); it != end(); it++)
        xml->push_back(it->second->toXML());

    return xml;
}
auto ChatRoomArray::toInvoke() const -> shared_ptr<Invoke>
{
    Invoke *invoke = new Invoke("handleRoomArray", toXML());
    return shared_ptr<Invoke>(invoke);
}
```

examples/chat_service/ChatRoom.cpp

```
#include "ChatRoom.hpp"
#include "ChatRoomArray.hpp"
#include "ChatMessage.hpp"

#include "ChatUser.hpp"
#include "ChatClient.hpp"
#include "ChatService.hpp"

#include <samchon/library/XML.hpp>
#include <samchon/protocol/Invoke.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;
using namespace samchon::example::chat_service;

/* ----------------------------------------------------------
    CONSTRUCTORS
---------------------------------------------------------- */
ChatRoom::ChatRoom(ChatRoomArray *roomArray, const string &name,
    ChatUser *host)
    : super(),
    IProtocol()
{
    this->roomArray = roomArray;

    this->name = name;
    this->host = host;
}

void ChatRoom::registerClient(ChatService *service)
{
    participants.insert(service);

    sendData(shared_ptr<Invoke>(new Invoke("handleRoom", toXML())));
    roomArray->notify();
}
void ChatRoom::eraseClient(ChatService *service)
{
    if (participants.erase(service) == 0)
        roomArray->erase(this->name);
    else
        sendData(shared_ptr<Invoke>(new Invoke("handleRoom", toXML())));

    roomArray->notify();
}

/* ----------------------------------------------------------
    CHAIN OF RESPONSIBILITY
---------------------------------------------------------- */
void ChatRoom::replyData(shared_ptr<Invoke> invoke)
{
    if(invoke->getListener() == "sendMessage")
    {
        shared_ptr<Invoke> ivk(new Invoke("handleMessage"));
```

```
        ivk->at(0) = invoke->at(0);

        sendData(ivk);
    }
}
void ChatRoom::sendData(shared_ptr<Invoke> invoke)
{
    UniqueReadLock uk(participants.get_allocator().getMutex());

    if(invoke->getListener() == "handleMessage")
    {
        ChatMessage message;
        message.construct(invoke->at(0)->getValueAsXML());

        if(message.getListener().empty() == true)
            for (auto it = participants.begin(); it != participants.end(); it++)
                (*it)->sendData(invoke);
        else
            for (auto it = participants.begin(); it != participants.end(); it++)
                if((*it)->getClient()->getUser()->getID() == message.getListener())
                    (*it)->sendData(invoke);
    }
    else
        for (auto it = participants.begin(); it != participants.end(); it++)
            (*it)->sendData(invoke);
}

/* ------------------------------------------------------------
    EXPORTS
------------------------------------------------------------ */
auto ChatRoom::toXML() const -> shared_ptr<XML>
{
    shared_ptr<XML> &xml = super::toXML();
    xml->setProperty("name", name);
    xml->setProperty("host", host->getID());

    for (auto it = participants.begin(); it != participants.end(); it++)
    {
        shared_ptr<XML> participant(new XML());
        participant->setTag("participant");
        participant->setValue((*it)->getClient()->getUser()->getID());

        xml->push_back(participant);
    }
    return xml;
}
```

examples/chat_service/ChatMessage.cpp

```
#include "ChatMessage.hpp"

#include <samchon/library/XML.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;
using namespace samchon::example::chat_service;

/* ------------------------------------------------------------
    CONSTRUCTORS
------------------------------------------------------------ */
ChatMessage::ChatMessage()
    : super()
{
}
void ChatMessage::construct(shared_ptr<XML> xml)
{
    this->orator = xml->getProperty("orator");
    this->message = xml->getProperty("message");

    if(xml->hasProperty("listener") == true)
        this->listener = xml->getProperty("listener");
    else
        this->listener.clear();
}

/* ------------------------------------------------------------
    GETTERS
------------------------------------------------------------ */
auto ChatMessage::getListener() const -> string
{
    return listener;
}
auto ChatMessage::toXML() const -> shared_ptr<XML>
{
    shared_ptr<XML> &xml = super::toXML();
```

```
    xml->setProperty("orator", orator);
    xml->setProperty("message", message);

    if(listener.empty() == false)
        xml->setProperty("listener", listener);
    return xml;
}

```

Author

Jeongho Nam

5.10 samchon::protocol::slave Namespace Reference

Package of external system, within the framework of slave.

Classes

- class ParallelClient

  A slave parallel client.

- class ParallelServer

  A slave parallel server.

- class ParallelSystem

  A slave parallel system.

- class SlaveClient

  A slave client.

- class SlaveServer

  A slave server.

- class SlaveSystem

  A slave system.

5.10.1 Detailed Description

Package of external system, within the framework of slave.

A package for composing distributed or parallel processing system in slave side.

Slave system get orders from master, processes the orders and report the result with its elapsed time for estimating performance of the slave system.

cpp/protocol_slave.png

Author

Jeongho Nam

Chapter 6

Class Documentation

6.1 Base64 Class Reference

Utility class for base64 format's en-decoding.

```
#include <Base64.hpp>
```

Static Public Member Functions

- static auto encode (const ByteArray &) -> std::string

  Encode from binary data to base64-string.

- static auto decode (const std::string &) -> ByteArray

  Decode from base64-string to binary data.

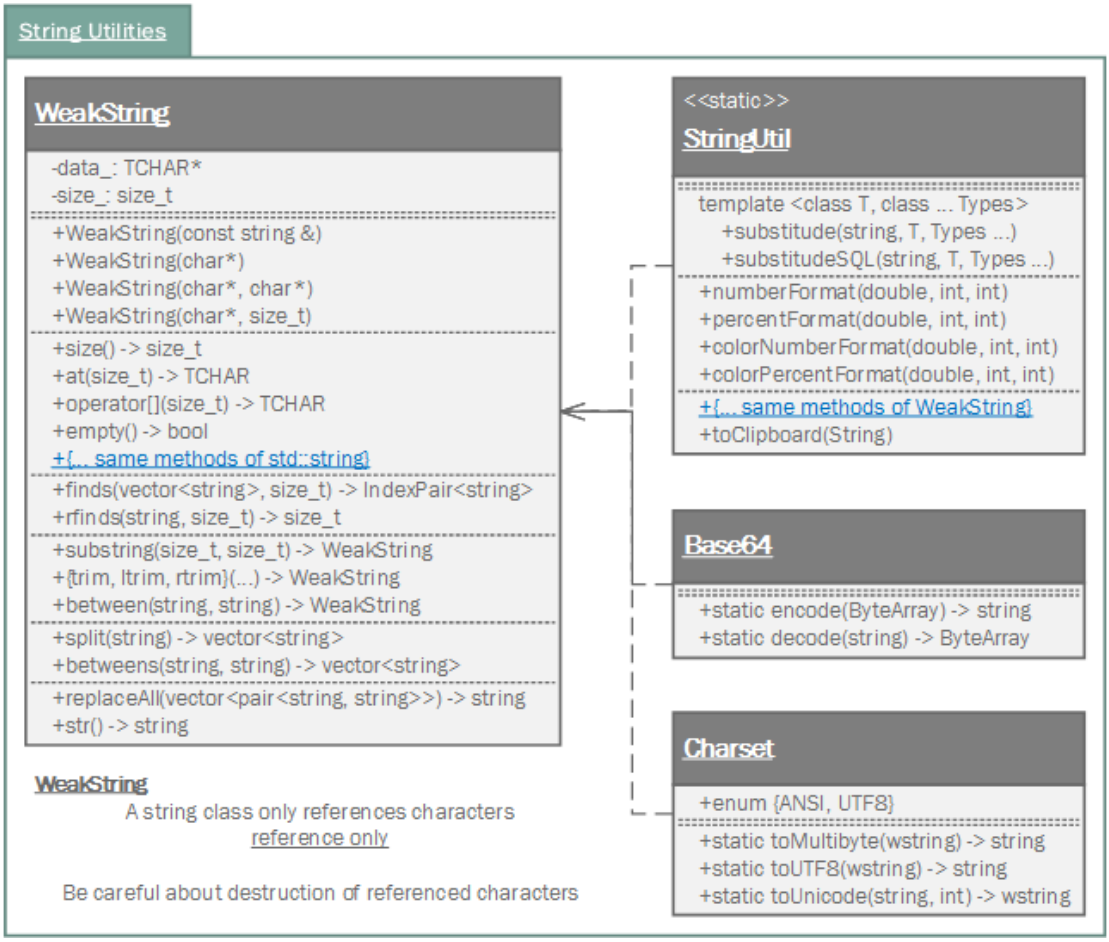
6.1.1 Detailed Description

Utility class for base64 format's en-decoding.

Base64 supports

- Encoding from binary data to base64-string

- Decoding from base64-string to binary data



See also

samchon::library

Author

Jeongho Nam

6.1.2 Member Function Documentation

6.1.2.1 auto encode (const ByteArray & *byteArray*) -> std::string [static]

Encode from binary data to base64-string.

Encodes a binary data to base64-string

Parameters

byteArray	binary data you want to encode

Returns

A base64-string encoded from binary data

6.1.2.2 auto decode (const std::string & *str*) -> ByteArray [static]

Decode from base64-string to binary data.

Decodes a base64 encoded string to binary data

Note

If the encoded string is not whole but a part of the binary data, Base64Decoder will be better

Parameters

str	A encoded string representing binary data as base64 format

Returns

A binary data decoded from string

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Base64.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Base64.cpp

6.2 ByteArray Class Reference

Binary data class.

```
#include <ByteArray.hpp>
```

Public Member Functions

- ByteArray ()

  Default Constructor.
- ByteArray (const ByteArray &)

  Copy Constructor.
- ByteArray (ByteArray &&)

  Move Constructor.
- auto getPosition () const -> size_t

  Get position.
- void setPosition (size_t)

  Set poisition.
- template<typename _Ty >
  auto read () const -> _Ty

  Read data.
- template<typename _Ty >
  auto readReversely () const -> _Ty

  Read a reversed data.
- template<typename _Ty >
  void write (const _Ty &val)

  Write a data.
- template<typename _Ty >
  void writeReversely (const _Ty &val)

  Write a data.
- auto compress () const -> ByteArray

  Compress the binary data

  .
- auto decompress () const -> ByteArray

  Decompress the binary data.

Static Public Member Functions

- template<typename _Ty >
  static auto reverse (const _Ty &val) -> _Ty

  Reverse byte ordering.

Private Attributes

- size_t position

  Current position of the ByteArray.

  .

6.2.1 Detailed Description

Binary data class.

ByteArray is the standard class for handling binary data in Samchon Framework.

ByteArray provides methods for reading and writing binary data.

In addition, zlib compression and decompression are supported.

Note

ByteArray is a class only for advanced developer who can access data on byte level as the reason of:

- Handling File Data
- Custom protocol for network I/O
- Optimizing the size of data for performance

If you're not trying to use ByteArray as the reason of what I've mentioned but only for representing entity data, I recomment you to use Entity instead.

See also

samchon::library

Author

Jeongho Nam

6.2.2 Member Function Documentation

6.2.2.1 auto getPosition ( ) const -> size_t

Get position.

Get a position represents starting point of bytes to read

6.2.2.2 void setPosition (size_t *val*)

Set poisition.

Set a position represents starting point of bytes to read

6.2.2.3 static auto reverse (const _Ty & *val*) -> _Ty [inline], [static]

Reverse byte ordering.

Creates a copy of data which of byte ordering is reversed.

- BIG_ENDIAN to SMALL_ENDIAN li SMALL_ENDIAN to BIG_ENDIAN

6.2.2.4 auto read ( ) const -> _Ty [inline]

Read data.

Reads a data(_Ty) from (starting) position of the ByteArray and adds the position following the size.

Template Parameters

_Ty	A type of data to be returned

Exceptions

exception	out of range

Returns

Read data from the byte stream

6.2.2.5 auto readReversely ( ) const -> _Ty [inline]

Read a reversed data.

Reads data(_Ty) from (starting) position of the ByteArray, which byte ordering is reversed, from the ByteArray and adds the position following the size.

- BIG_ENDIAN to SMALL_ENDIAN li SMALL_ENDIAN to BIG_ENDIAN

Template Parameters

_Ty	A type of data to be returned

Exceptions

exception	out of range

Returns

Read data from the byte stream

6.2.2.6 void write (const _Ty & *val*) [inline]

Write a data.

Writes a data(_Ty) to tail of the ByteArray

Template Parameters

_Ty	A type of data to write

Parameters

val	Something to write on ByteArray

6.2.2.7 void writeReversely (const _Ty & *val*) [inline]

Write a data.

Writes a data(_Ty), which byte order is reversed, to tail of the ByteArray

- BIG_ENDIAN to SMALL_ENDIAN li SMALL_ENDIAN to BIG_ENDIAN

Template Parameters

_Ty	A type of data to write

Parameters

val	Something to write on ByteArray

6.2.2.8 auto compress ( ) const -> ByteArray

Compress the binary data

.

Generates a binary data compressed from the ByteArray.

The binary data will be compressed by zlib library.

Returns

ByteArray which is compressed

6.2.2.9 auto decompress ( ) const -> ByteArray

Decompress the binary data.

Generates a decompressed binary data form the ByteArray.

The binary data will be decompressed by zlib library.

Returns

ByteArray that is decompressed

6.2.3 Member Data Documentation

6.2.3.1 size_t position [private]

Current position of the ByteArray.

.

If you call read, starting pointer will be the position

- (_Ty *ptr* = *(_Ty)*data() + position)

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/ByteArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/ByteArray.cpp

6.3 CaseGenerator Class Reference

Case generator.

```
#include <CaseGenerator.hpp>
```

Public Member Functions

- CaseGenerator (size_t n, size_t r)

  Construct from size of N and R.
- auto size () const -> size_t

  Get size of all cases.
- virtual auto operator[ ] (size_t) const -> std::vector< size_t >=0

  Get x'th case.
- auto at (size_t) const -> std::vector< size_t >
- auto n () const -> size_t

  Get size of the N.
- auto r () const -> size_t

  Get size of the R.
- auto toMatrix () const -> std::vector< std::vector< size_t >>

  Create a matrix containing all cases.

Protected Attributes

- size_t n_

  N, size of the candidates.
- size_t r_

  R, size of elements of each case.
- size_t size_

  Size, the number of all cases.

6.3.1 Detailed Description

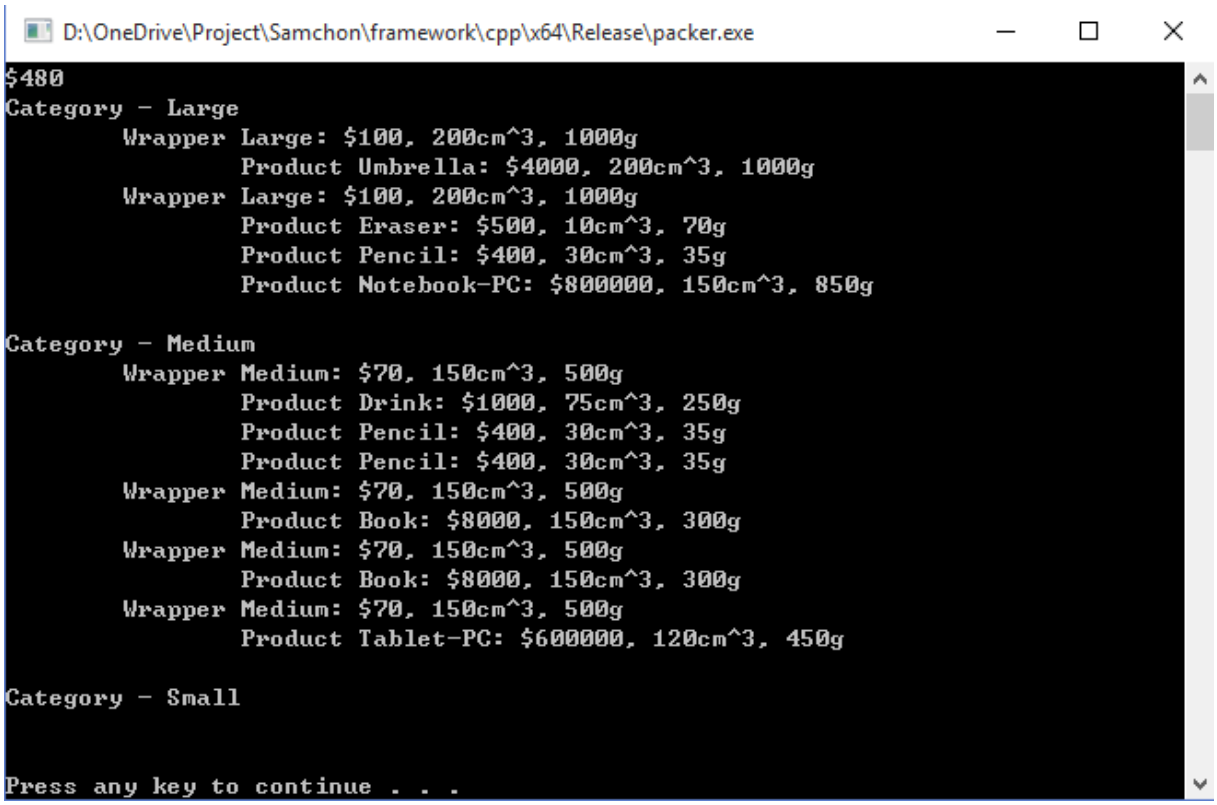
Case generator.

An abstract case generator using like a matrix

- nTTr(n^r) -> CombinedPermutationGenerator
- nPr -> PermutationGenerator
- n! -> FactorialGenerator

Example source

Packer - Find the best packaging solution



examples/packer/Packer.hpp

```
#pragma once
#include <vector>
#include <memory>
#include <string>

namespace samchon
{
    namespace example
    {
        namespace packer
        {
            class WrapperArray;
            class Wrapper;
            class Product;

            class Packer
                : public std::vector<std::shared_ptr<WrapperArray>>
            {
            private:
                typedef std::vector<std::shared_ptr<WrapperArray>> super;

                std::vector<Product> *productArray;

                std::vector<Wrapper> *wrapperArray;

            public:
                /* ---------------------------------------------------------
                    CONSTRUCTORS
                --------------------------------------------------------- */
                Packer(std::vector<Product> *, std::vector<Wrapper> *);

                Packer(const Packer &);

                /* ---------------------------------------------------------
                    CALCULATE AND OPTIMIZE
                --------------------------------------------------------- */
                void optimize();

                auto calcPrice() const -> int;

```

```
                /* -------------------------------------------------------
                    EXPORT
                ------------------------------------------------------- */
                auto toString() const -> std::string;
            };
        };
    };
};
```

examples/packer/WrapperArray.hpp

```
#pragma once
#include <vector>
#include <memory>

namespace samchon
{
    namespace example
    {
        namespace packer
        {
            class Wrapper;
            class Product;

            class WrapperArray
                : private std::vector<std::shared_ptr<Wrapper>>
            {
            private:
                typedef std::vector<std::shared_ptr<Wrapper>> super;

                std::vector<Product*> reserved;

                Wrapper *sample;

            public:
                /* -------------------------------------------------------
                    CONSTRUCTORS
                ------------------------------------------------------- */
                WrapperArray(Wrapper *);

                auto tryInsert(Product*) -> bool;

                /* -------------------------------------------------------
                    CALCULATE AND OPTIMIZE
                ------------------------------------------------------- */
                void optimize();

                auto calcPrice() const -> int;

                /* -------------------------------------------------------
                    EXPORT
                ------------------------------------------------------- */
                auto toString() const -> std::string;
            };
        };
    };
};
```

examples/packer/Wrapper.hpp

```
#pragma once
#include <vector>
#include "Instance.hpp"

namespace samchon
{
    namespace example
    {
        namespace packer
        {
            class Product;

            class Wrapper
                : public Instance,
                private std::vector<Product*>
            {
            private:
                typedef Instance super;

            public:
                /* -------------------------------------------------------
                    CONSTRUCTORS
                ------------------------------------------------------- */
                Wrapper(const std::string &name, int price, int volume, int weight);

```

```
49                  Wrapper(const Wrapper &wrapper);
50                  virtual ~Wrapper() = default;
51
62                  auto tryInsert(Product *) -> bool;
63
64                  /* ----------------------------------------------------------
65                      EXPORT
66                  ---------------------------------------------------------- */
73                  virtual auto toString() const -> std::string override;
74              };
75          };
76      };
77 };
```

examples/packer/Product.hpp

```
1 #pragma once
2 #include "Instance.hpp"
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace packer
9         {
21              class Product
22                  : public Instance
23              {
24              private:
25                  typedef Instance super;
26
27              public:
28                  /* ----------------------------------------------------------
29                      CONSTRUCTOR
30                  ---------------------------------------------------------- */
39                  Product(const std::string &name, int price, int volume, int weight);
40                  virtual ~Product() = default;
41
42                  /* ----------------------------------------------------------
43                      EXPORT
44                  ---------------------------------------------------------- */
48                  virtual auto toString() const -> std::string override;
49              };
50          };
51      };
52 };
```

examples/packer/Instance.hpp

```
1 #pragma once
2 #include <string>
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace packer
9         {
21              class Instance
22              {
23              protected:
27                  std::string name;
28
32                  int price;
33
37                  int volume;
38
42                  int weight;
43
44              public:
45                  /* ----------------------------------------------------------
46                      CONSTRUCTORS
47                  ---------------------------------------------------------- */
56                  Instance(const std::string &name, int price, int volume, int weight);
57                  virtual ~Instance() = default;
58
59                  /* ----------------------------------------------------------
60                      GETTERS
61                  ---------------------------------------------------------- */
65                  auto getName() const -> std::string;
66
70                  auto getPrice() const -> int;
71
75                  auto getVolume() const -> int;
76
```

```
                auto getWeight() const -> int;

                /* -------------------------------------------------------
                    EXPORT
                ------------------------------------------------------- */
                virtual auto toString() const -> std::string;
            };
        };
    };
};
```

examples/packer/main.cpp

```
#include <iostream>
#include <vector>

#include "Packer.hpp"
#   include "Product.hpp"
#   include "Wrapper.hpp"

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace std;
using namespace samchon::example::packer;

void main()
{
    vector<Product> productArray =
    {

        Product("Eraser", 500, 10, 70),
        Product("Pencil", 400, 30, 35),
        Product("Pencil", 400, 30, 35),
        Product("Pencil", 400, 30, 35),
        Product("Book", 8000, 150, 300),
        Product("Book", 8000, 150, 300),
        Product("Drink", 1000, 75, 250),
        Product("Umbrella", 4000, 200, 1000),
        Product("Notebook-PC", 800000, 150, 850),
        Product("Tablet-PC", 600000, 120, 450)
    };
    vector<Wrapper> wrapperArray =
    {
        Wrapper("Large", 100, 200, 1000),
        Wrapper("Medium", 70, 150, 500),
        Wrapper("Small", 50, 100, 250)
    };

    Packer packer(&productArray, &wrapperArray);
    packer.optimize();

    cout << packer.toString() << endl;
    system("pause");
}
```

examples/packer/Packer.cpp

```
#include "Packer.hpp"

#include "WrapperArray.hpp"
#include "Wrapper.hpp"
#include "Product.hpp"

#include <samchon/library/CombinedPermutationGenerator.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::example::packer;

Packer::Packer(vector<Product> *productArray, vector<Wrapper> *wrapperArray)
    : super()
{
    this->productArray = productArray;
```

```
    this->wrapperArray = wrapperArray;

    for(size_t i = 0; i < wrapperArray->size(); i++)
        emplace_back(new WrapperArray(&wrapperArray->at(i)));
}
Packer::Packer(const Packer &packer)
    : Packer(packer.productArray, packer.wrapperArray)
{
}

/* ------------------------------------------------------
    CALCULATE AND OPTIMIZE
------------------------------------------------------ */
void Packer::optimize()
{
    if(empty() == true || productArray->empty() == true)
        return;

    CombinedPermutationGenerator caseGenerator(size(), productArray->size());
    shared_ptr<Packer> minPacker = nullptr;

    for (size_t i = 0; i < caseGenerator.size(); i++)
    {
        vector<size_t> &row = caseGenerator[i];
        shared_ptr<Packer> packer(new Packer(*this));
        bool validity = true;

        for (size_t j = 0; j < row.size(); j++)
        {
            Product *product = &productArray->at(j);
            shared_ptr<WrapperArray> &wrapperArray = packer->at( row[j] );

            if (wrapperArray->tryInsert(product) == false)
            {
                validity = false;
                break;
            }
        }

        if(validity == false)
            continue;

        //OPTIMIZE ALL WRAPPERS IN A PACKER
        for (size_t j = 0; j < packer->size(); j++)
            packer->at(j)->optimize();

        if (minPacker == nullptr ||
            packer->calcPrice() < minPacker->calcPrice())
        {
            minPacker = packer;
        }
    }

    //COPY
    assign(minPacker->begin(), minPacker->end());
}
auto Packer::calcPrice() const -> int
{
    int price = 0;
    for (size_t i = 0; i < size(); i++)
        price += at(i)->calcPrice();

    return price;
}

/* ------------------------------------------------------
    EXPORT
------------------------------------------------------ */
auto Packer::toString() const -> string
{
    string str = "$" + to_string(calcPrice()) + "\n";
    for (int i = 0; i < size(); i++)
        str += at(i)->toString() + "\n";

    return move(str);
}
```

examples/packer/WrapperArray.cpp

```
#include "WrapperArray.hpp"
#   include "Wrapper.hpp"
#   include "Product.hpp"

#include <samchon/library/FactorialGenerator.hpp>
#include <mutex>

```

```
using namespace std;
using namespace samchon::library;
using namespace samchon::example::packer;

/* ---------------------------------------------------------
    CONSTRUCTOR
--------------------------------------------------------- */
WrapperArray::WrapperArray(Wrapper *sample)
    : super()
{
    this->sample = sample;
}

/* ---------------------------------------------------------
    CALCULATE AND OPTIMIZE
--------------------------------------------------------- */
auto WrapperArray::tryInsert(Product *product) -> bool
{
    if (product->getVolume() > sample->getVolume() ||
        product->getWeight() > sample->getWeight())
    {
        return false;
    }

    reserved.push_back(product);
    return true;
}
void WrapperArray::optimize()
{
    if(reserved.empty() == true)
        return;

    FactorialGenerator factorial(reserved.size());
    shared_ptr<WrapperArray> minWrapperArray = nullptr;

    mutex mtx;

    //#pragma omp parallel for
    for (int i = 0; i < factorial.size(); i++)
    {
        shared_ptr<WrapperArray> wrapperArray(new WrapperArray(this->sample));
        vector<size_t> &row = factorial[i];

        for (size_t j = 0; j < row.size(); j++)
        {
            Product *product = this->reserved[row[j]];

            if (wrapperArray->empty() == true ||
                wrapperArray->at(wrapperArray->size() - 1)->tryInsert(product) == false)
            {
                Wrapper *wrapper = new Wrapper(*this->sample);
                wrapper->tryInsert(product);

                wrapperArray->emplace_back(wrapper);
            }
        }

        //unique_lock<mutex> uk(mtx);
        if (minWrapperArray == nullptr ||
            wrapperArray->size() < minWrapperArray->size())
        {
            minWrapperArray = wrapperArray;
        }
    }

    assign(minWrapperArray->begin(), minWrapperArray->end());
}
auto WrapperArray::calcPrice() const -> int
{
    return sample->getPrice() * (int)size();
}

/* ---------------------------------------------------------
    EXPORT
--------------------------------------------------------- */
auto WrapperArray::toString() const -> string
{
    string str = "Category - " + sample->getName() + "\n";
    for (size_t i = 0; i < size(); i++)
        str += at(i)->toString() + "\n";

    return move(str);
}
```

examples/packer/Wrapper.cpp

```
#include "Wrapper.hpp"
#   include "Product.hpp"

using namespace std;
using namespace samchon::example::packer;

/* ---------------------------------------------------------
    CONSTRUCTORS
--------------------------------------------------------- */
Wrapper::Wrapper(const string &name, int price, int volume, int weight)
    : super(name, price, volume, weight),
    vector<Product*>()
{
}
Wrapper::Wrapper(const Wrapper &wrapper)
    : super(wrapper),
    vector<Product*>()
{
}

auto Wrapper::tryInsert(Product *product) -> bool
{
    int volume = 0;
    int weight = 0;

    for (size_t i = 0; i < size(); i++)
    {
        volume += at(i)->getVolume();
        weight += at(i)->getWeight();
    }

    if (product->getVolume() + volume > this->volume ||
        product->getWeight() + weight > this->weight)
    {
        return false;
    }

    push_back(product);
    return true;
}

/* ---------------------------------------------------------
    EXPORT
--------------------------------------------------------- */
auto Wrapper::toString() const -> string
{
    string str = "\tWrapper " + super::toString() + "\n";
    for (size_t i = 0; i < size(); i++)
        str += "\t\t" + at(i)->toString() + ((i == size() - 1) ? "" : "\n");

    return move(str);
};
```

examples/packer/Product.cpp

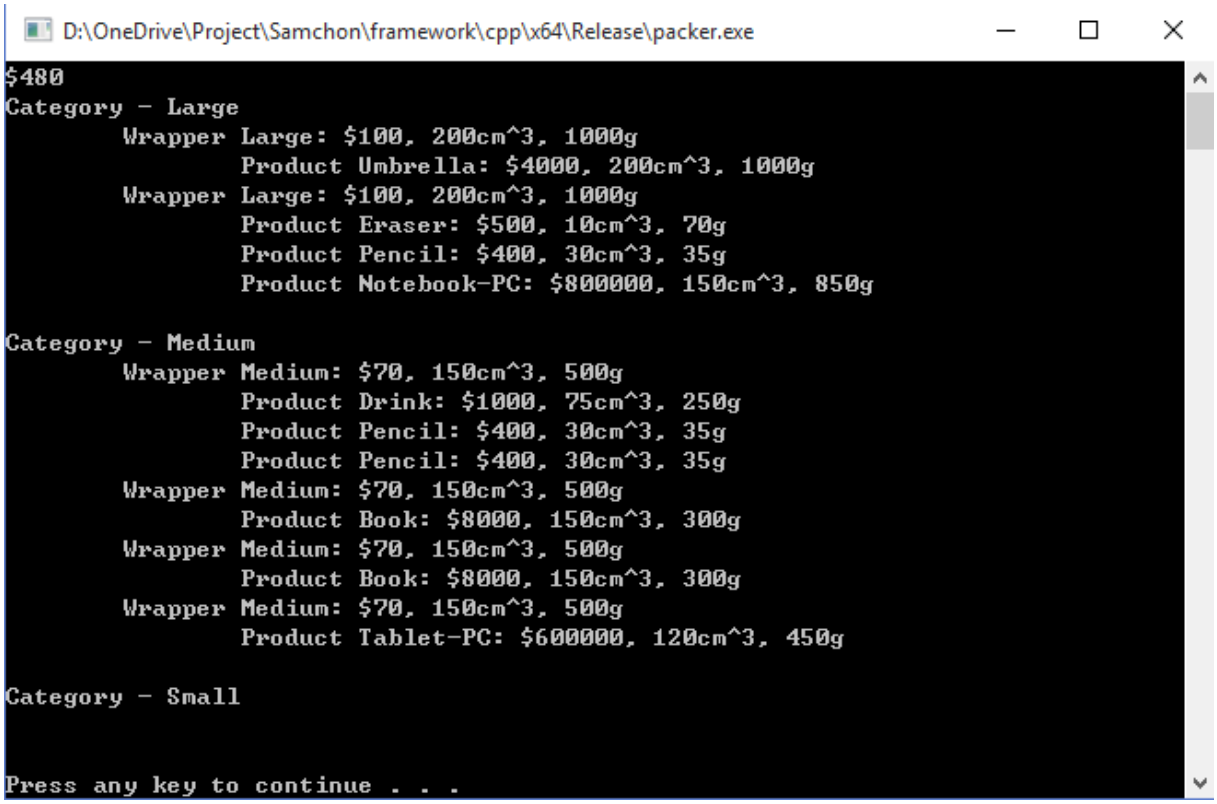
```
#pragma once
#include "Product.hpp"

using namespace std;
using namespace samchon::example::packer;

Product::Product(const string &name, int price, int volume, int weight)
    : super(name, price, volume, weight)
{
}
auto Product::toString() const -> string
{
    return "Product " + super::toString();
}
```

examples/packer/Instance.cpp

```
#include "Instance.hpp"

#include <samchon/library/StringUtil.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::example::packer;

/* ---------------------------------------------------------
    CONSTRUCTORS
--------------------------------------------------------- */
```

```
Instance::Instance(const string &name, int price, int volume, int weight)
{
    this->name = name;
    this->price = price;
    this->volume = volume;
    this->weight = weight;
}

/* -------------------------------------------------------
    GETTERS
------------------------------------------------------- */
auto Instance::getName() const -> string
{
    return name;
}
auto Instance::getPrice() const -> int
{
    return price;
}
auto Instance::getVolume() const -> int
{
    return volume;
}
auto Instance::getWeight() const -> int
{
    return weight;
}

/* -------------------------------------------------------
    EXPORT
------------------------------------------------------- */
auto Instance::toString() const -> string
{
    return StringUtil::substitute
        (
            "{1}: ${2}, {3}cm^3, {4}g",
            name, price, volume, weight
        );
}
```

Result of the example



See also

samchon::library

Author

Jeongho Nam

6.3.2 Constructor & Destructor Documentation

6.3.2.1 CaseGenerator (size_t *n*, size_t *r*)

Construct from size of N and R.

Parameters

n	Size of candidates
r	Size of elements of each case

6.3.3 Member Function Documentation

6.3.3.1 auto size ( ) const -> size_t

Get size of all cases.

Returns

Get a number of the all cases

6.3.3.2 virtual auto operator[ ] (size_t) const -> std::vector< size_t > [pure virtual]

Get x'th case.

Returns

The row of the x'th in combined permuation case

Implemented in CombinedPermutationGenerator, and PermutationGenerator.

6.3.3.3 auto at (size_t *index*) const -> std::vector<size_t>

CaseGenerator::operator[ ]()

6.3.3.4 auto toMatrix ( ) const -> std::vector<std::vector<size_t>>

Create a matrix containing all cases.

Warning

Size of n or r is enourmouse, then stack overflow can be occured.

Returns

A matrix containing all cases.

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/CaseGenerator.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/CaseGenerator.cpp

6.4 Charset Class Reference

A utility class supporting conversion between multiple character-sets.

```
#include <Charset.hpp>
```

Static Public Member Functions

- static auto toMultibyte (const std::string &) -> std::string

  Convert utf-8 to multibyte.

- static auto toMultibyte (const std::wstring &) -> std::string

  Convert unicode to multibyte.

- static auto toUTF8 (const std::string &) -> std::string

  Convert multibyte to utf-8.

- static auto toUTF8 (const std::wstring &) -> std::string

  Convert unicode to utf-8.

- static auto toUnicode (const std::string &, int) -> std::wstring

  Convert multibyte or utf-8 to unicode.

6.4.1 Detailed Description

A utility class supporting conversion between multiple character-sets.

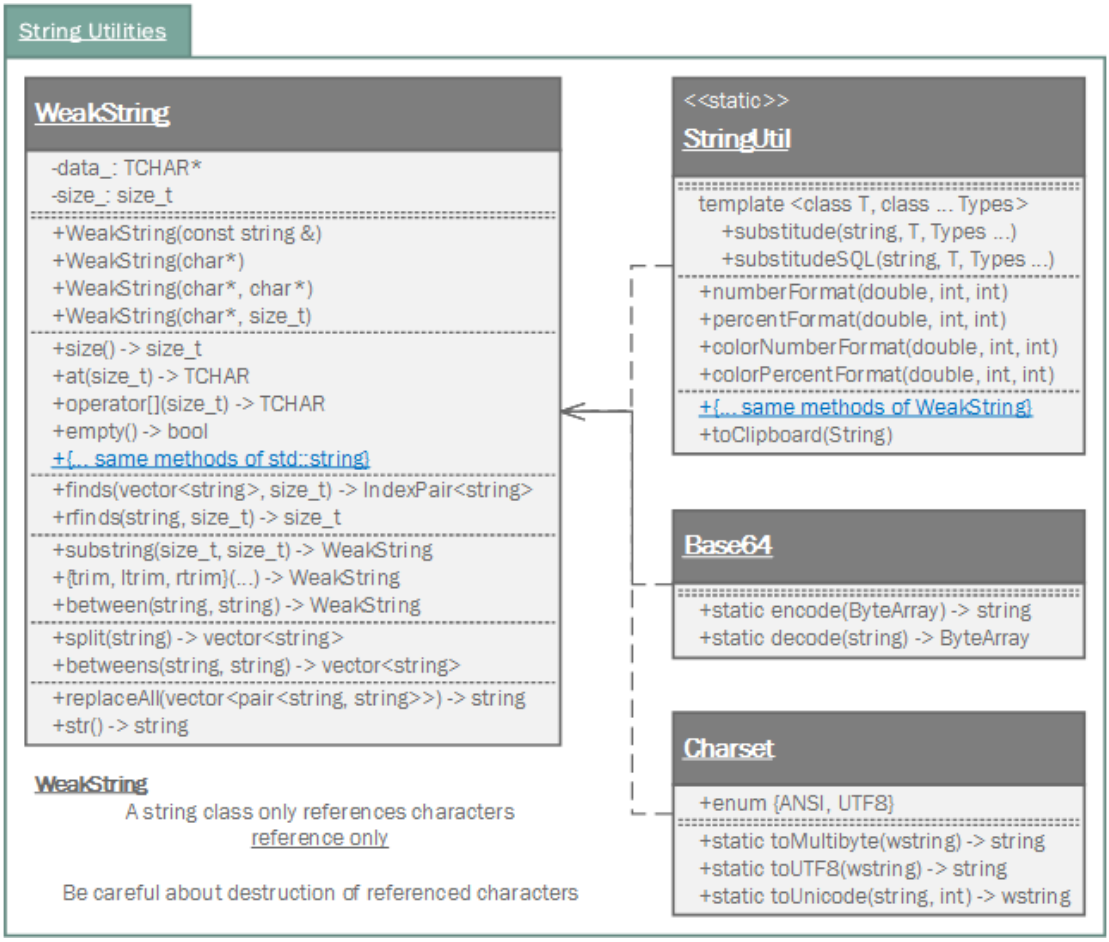
Charset class, it's easier to think iconv in linux

Supported character-sets

- Multibyte (ANSI)

- UTF-8

- Unicode (UTF-16)



Warning

In some system, std::wstring is not for utf-8 but for unicode (utf-16).

See also

samchon::library

Author

Jeongho Nam

6.4.2 Member Function Documentation

6.4.2.1 static auto toMultibyte (const std::string &) -> std::string [static]

Convert utf-8 to multibyte.

Converts utf-8 string to multibyte string.

If the utf-8 string has header int the front, it will be erased.

Warning

If utf-8 header is not at the front of string, it can't be erased and may cause breakage on letters

Parameters

str	A utf-8 string would be converted to multibyte

Returns

A multibyte string

6.4.2.2 static auto toMultibyte (const std::wstring &) -> std::string [static]

Convert unicode to multibyte.

Converts unicode string to multibyte string
If the unicode string has header in the front, it will be erased

Warning

If unicode header is not at the front of string, it can't be erased and may cause breakage on letters

Parameters

str	A unicode string to be converted

Returns

A multibyte string

6.4.2.3 static auto toUTF8 (const std::string &) -> std::string [static]

Convert multibyte to utf-8.

Converts multibyte string to utf-8 string
If the inputted string is not multibyte, then may cause breakage on letters

Warning

Converted utf-8 string does not contain the header

Parameters

str	A multibyte string would be converted

Returns

A utf-8 string

6.4.2.4 static auto toUTF8 (const std::wstring &) -> std::string [static]

Convert unicode to utf-8.

Converts unicode string to utf-8 string
Generated utf-8 string follows unicode string wheter to have header or not

- If the unicode string has header on first, utf-8 will also have the header, too
- If not, generated utf-8 string doesn't have the header, either

Parameters

str	A unicode string would be converted

Returns

A utf-8 string

6.4.2.5 static auto toUnicode (const std::string & , int) -> std::wstring [static]

Convert multibyte or utf-8 to unicode.

Parameters

str	A multibyte or utf-8 string would be converted
Designate	str is multibyte or utf-8 string

Returns

A unicode string

The documentation for this class was generated from the following file:

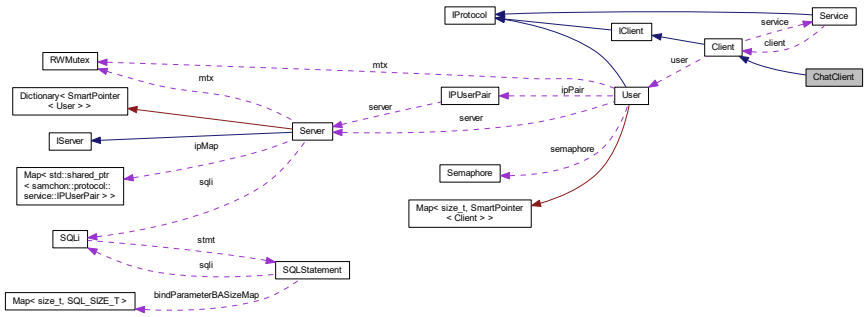
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Charset.hpp

6.5 ChatClient Class Reference

A client.

`#include <ChatClient.hpp>`

Collaboration diagram for ChatClient:



Public Member Functions

- ChatClient (protocol::service::User *)

  Construct from an user.

Protected Member Functions

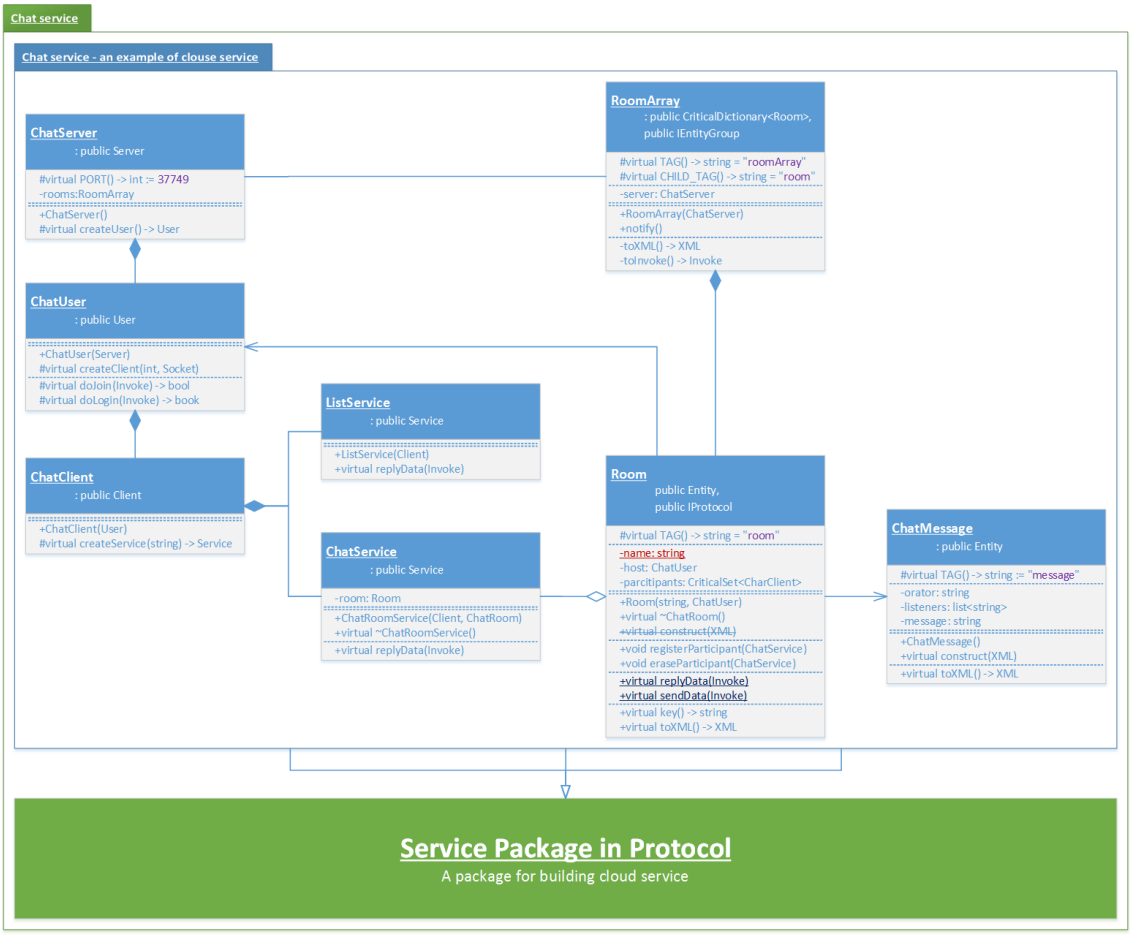
- auto createService (const std::string &) -> protocol::service::Service *override

  Factory method of Service.

Additional Inherited Members

6.5.1 Detailed Description

A client.



Author

Jeongho Nam

6.5.2 Member Function Documentation

6.5.2.1 auto createService (const std::string &) -> protocol::service::Service *override [override], [protected], [virtual]

Factory method of Service.

Parameters

name	Requested name representing a Service from the (physical) client

Returns

A new Service belongs the the Client

Implements Client.

The documentation for this class was generated from the following file:

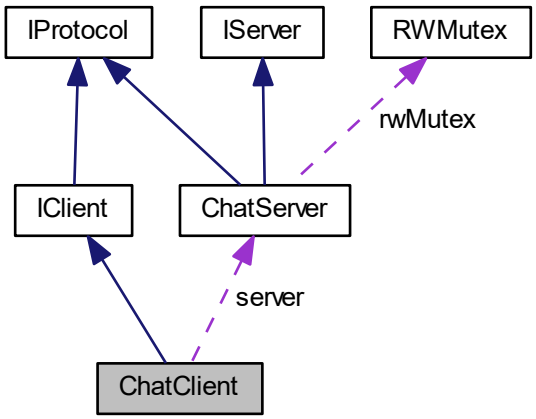
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatClient.hpp

6.6 ChatClient Class Reference

A chat client.

`#include <ChatClient.hpp>`

Collaboration diagram for ChatClient:



Public Member Functions

- ChatClient (ChatServer *, protocol::Socket *)

  Construct from a server and socket.
- virtual ~ChatClient ()

  Default Destructor.
- virtual void replyData (std::shared_ptr< protocol::Invoke >) override

  Shift responsibility to related chain.

Private Attributes

- ChatServer * server

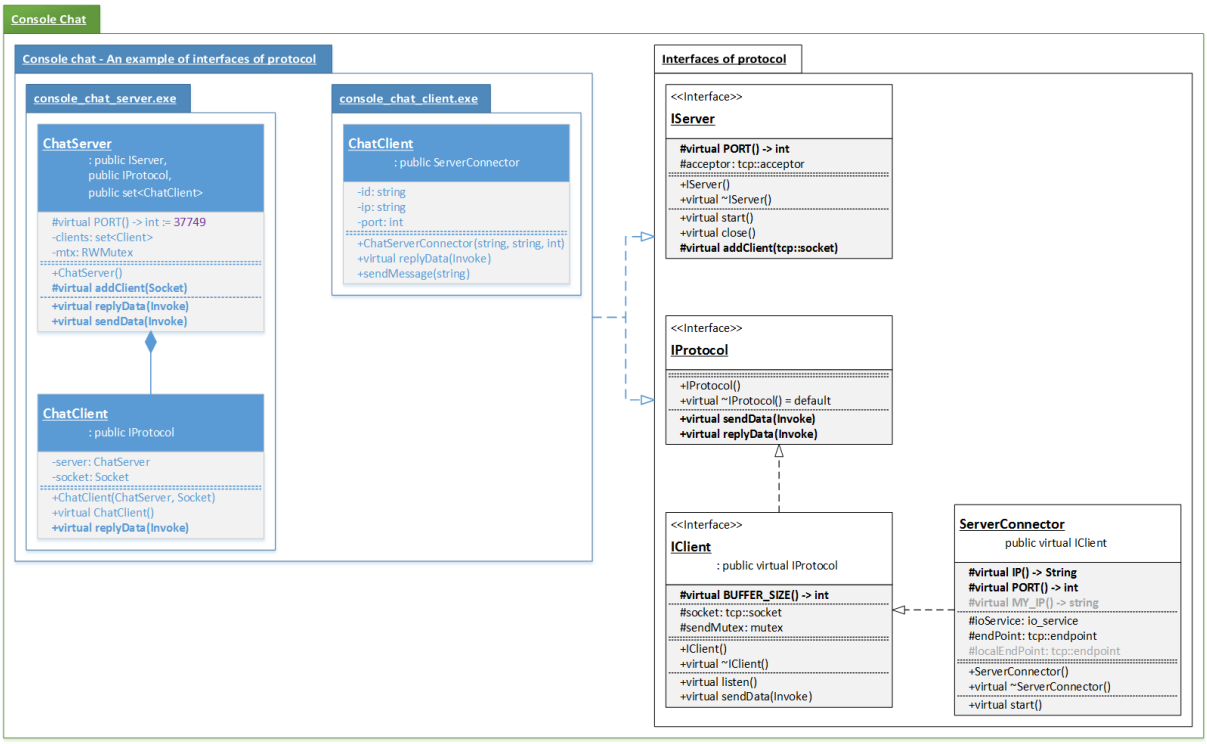
  A ChatServer that the ChatClient is belonged to.

Additional Inherited Members

6.6.1 Detailed Description

A chat client.

A simple chat client belonged to a chat server



Author

Jeongho Nam

6.6.2 Constructor & Destructor Documentation

6.6.2.1 ChatClient (ChatServer * *server*, protocol::Socket * *socket*)

Construct from a server and socket.

Parameters

server	A server the client is belonged to.
socket	A socket interacting with physical client.

6.6.2.2 ~ChatClient ( ) [virtual]

Default Destructor.

Erase from ChatServer::clientSet.

6.6.3 Member Function Documentation

6.6.3.1 void replyData (std::shared_ptr< protocol::Invoke >) [override], [virtual]

Shift responsibility to related chain.

Call ChatServer::replyData().

The documentation for this class was generated from the following files:

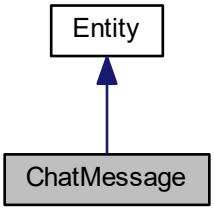
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/console_chat_server/ChatClient.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/console_chat_server/ChatClient.cpp

6.7 ChatMessage Class Reference

A chat message.

`#include <ChatMessage.hpp>`

Collaboration diagram for ChatMessage:



Public Member Functions

- ChatMessage ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >)

  Construct data of the Entity from an XML object.
- auto getListener () const -> std::string

  Get listener.
- virtual auto toXML () const -> std::shared_ptr< library::XML >

  Get an XML object represents the Entity.

Protected Member Functions

- virtual auto TAG () const -> std::string

  A tag name when represented by XML.

Private Attributes

- std::string orator

  An orator, who told.
- std::string listener

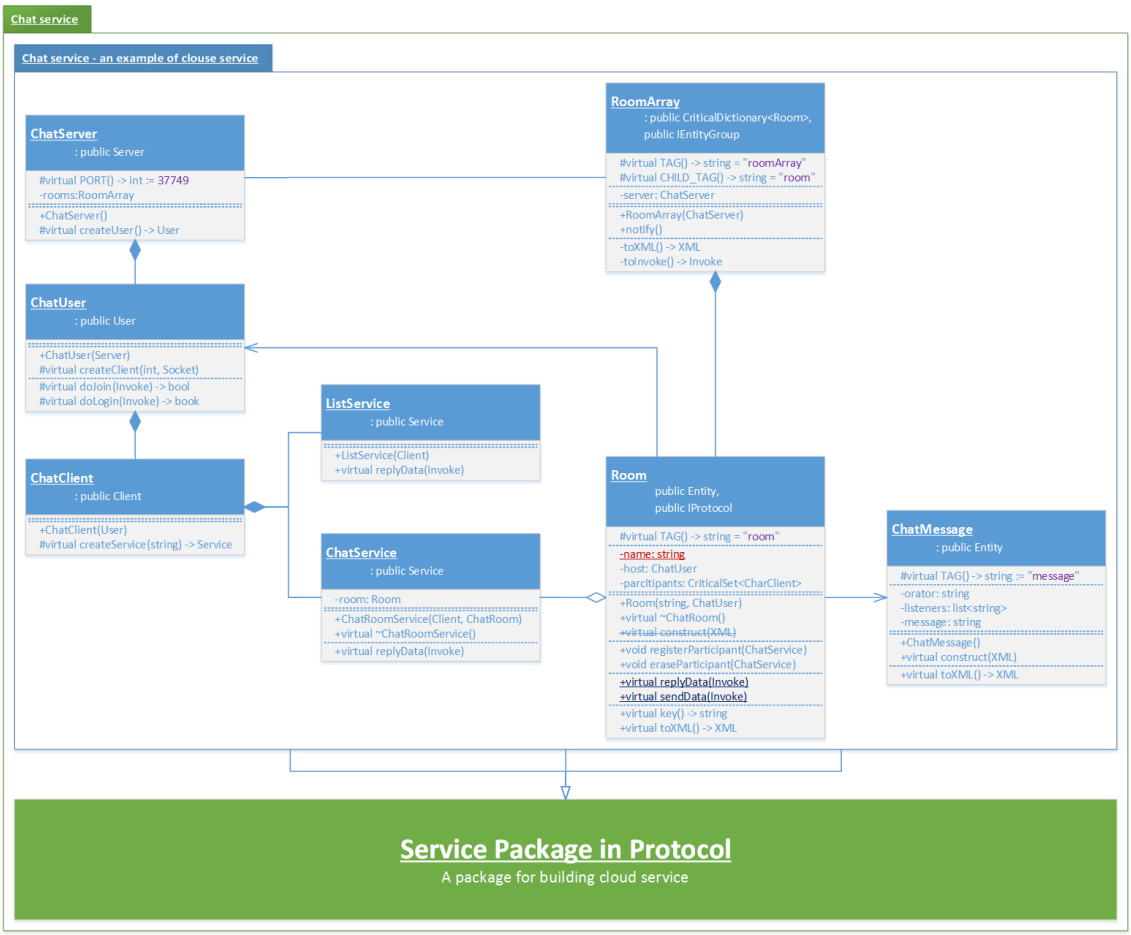
  A listener, target of whispering.
- std::string message

  A message, what to say.

6.7.1 Detailed Description

A chat message.

An entity class representing a chat message.



Author

Jeongho Nam

6.7.2 Member Function Documentation

6.7.2.1 virtual auto TAG ( ) const -> std::string [inline], [protected], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.7.2.2 void construct (std::shared_ptr< library::XML >) [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

6.7.2.3 auto getListener ( ) const -> std::string

Get listener.

Used to determine which (physical) client will get the message.

6.7.2.4 auto toXML ( ) const -> std::shared_ptr<library::XML> [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

6.7.3 Member Data Documentation

6.7.3.1 std::string listener [private]

A listener, target of whispering.

If there's not a listener to whisper, the value is "ALL".

The documentation for this class was generated from the following files:

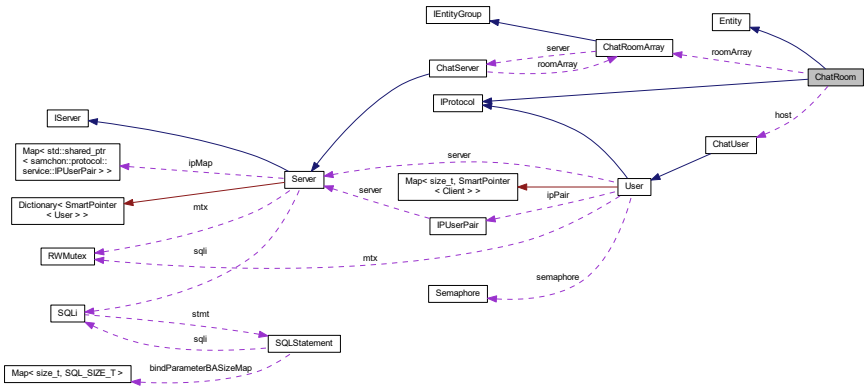
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatMessage.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatMessage.cpp

6.8 ChatRoom Class Reference

A chat room.

```
#include <ChatRoom.hpp>
```

Collaboration diagram for ChatRoom:



Public Member Functions

- ChatRoom (ChatRoomArray ∗, const std::string &, ChatUser ∗)

  Construct from rooms, name and host.
- void registerClient (ChatService ∗)

  Register a participant client.
- void eraseClient (ChatService ∗)

  Erase a participant client.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the Entity.

Protected Member Functions

- virtual auto TAG () const -> std::string

  A tag name when represented by XML.

Private Attributes

- ChatRoomArray ∗ roomArray

  All rooms that the room is belonged to.
- std::string name

  Name of the room.
- ChatUser ∗ host

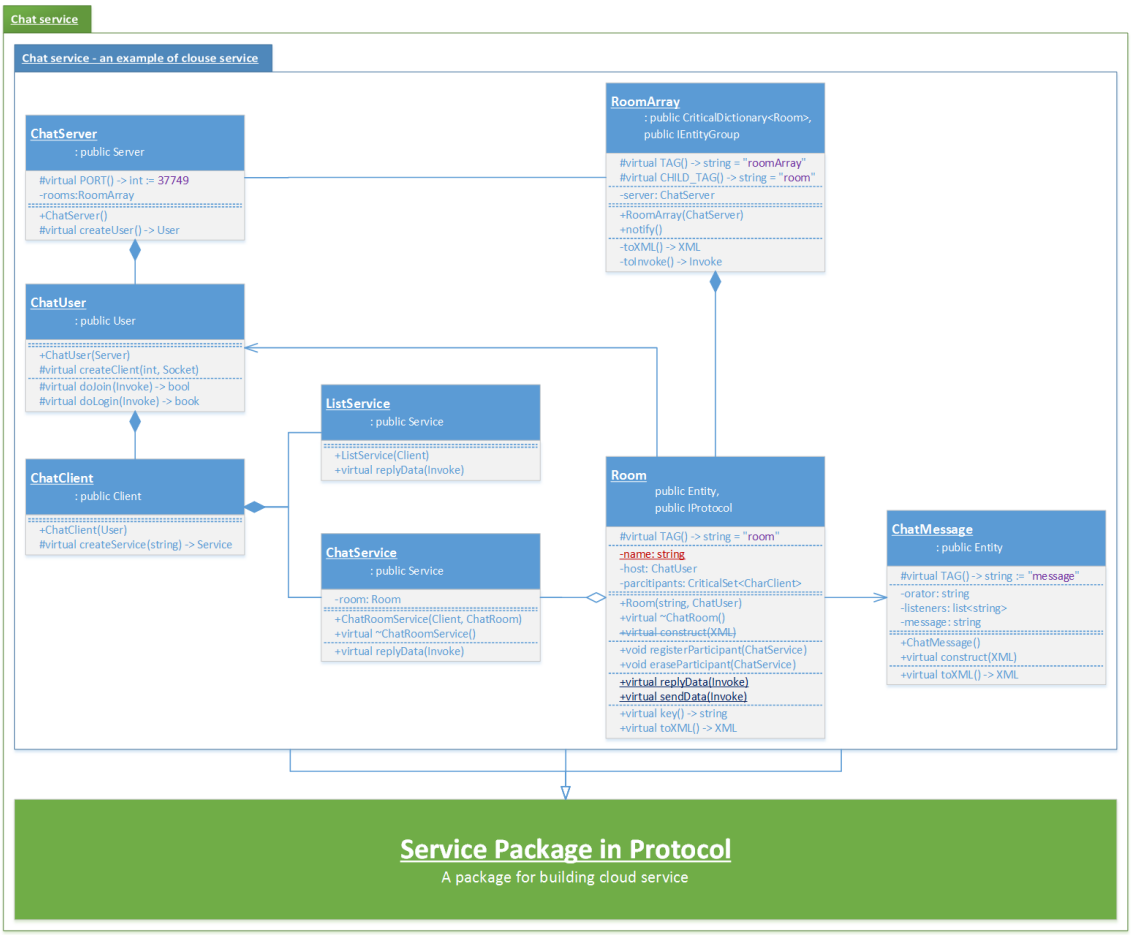
  A host, who established the room.
- library::CriticalSet< ChatService ∗ > participants

  Participants, ChatUser(s) in the room.

6.8.1 Detailed Description

A chat room.

An entity class representing a chat room containing participants. The ChatRoom class takes a responsibility of Invoke message chain between clients (participating ChatUser(s)) objects.



Author

Jeongho Nam

6.8.2 Constructor & Destructor Documentation

6.8.2.1 ChatRoom (ChatRoomArray * , const std::string & , ChatUser *)

Construct from rooms, name and host.

Parameters

rooms	Rooms managing ChatRoom(s).
name	Name of the room, the identifier.
host	A host who established the room.

6.8.3 Member Function Documentation

6.8.3.1 virtual auto TAG ( ) const -> std::string [inline], [protected], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.8.3.2 void registerClient (ChatService * *service*)

Register a participant client.

Registers a participant and notify the participation to another clients.

6.8.3.3 void eraseClient (ChatService * *service*)

Erase a participant client.

Erases a participant and notify others about the exiting.

6.8.3.4 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

6.8.4 Member Data Documentation

6.8.4.1 std::string name [private]

Name of the room.

The name is the identifier of the ChatRoom.

The documentation for this class was generated from the following files:

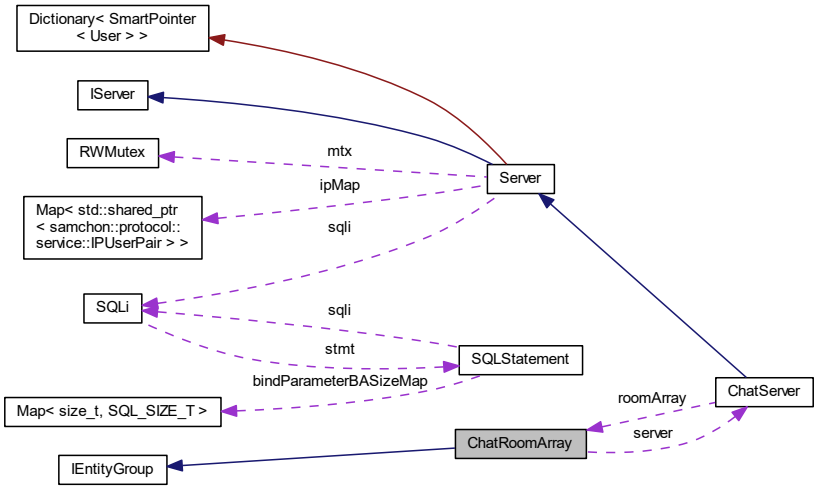
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatRoom.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatRoom.cpp

6.9 ChatRoomArray Class Reference

A manager of chat rooms.

```
#include <ChatRoomArray.hpp>
```

Collaboration diagram for ChatRoomArray:



Public Member Functions

- ChatRoomArray (ChatServer *)

  Construct from a chat server.
- void notify ()

  Notify changes of chat rooms.

Protected Member Functions

- virtual auto CHILD_TAG () const -> std::string

  A tag name of children.

Private Member Functions

- auto toXML () const -> std::shared_ptr< library::XML >

  Get an XML object representing the ChatRoomArray.
- auto toInvoke () const -> std::shared_ptr< protocol::Invoke >

  Get an Invoke message used to notifying change.

Private Attributes

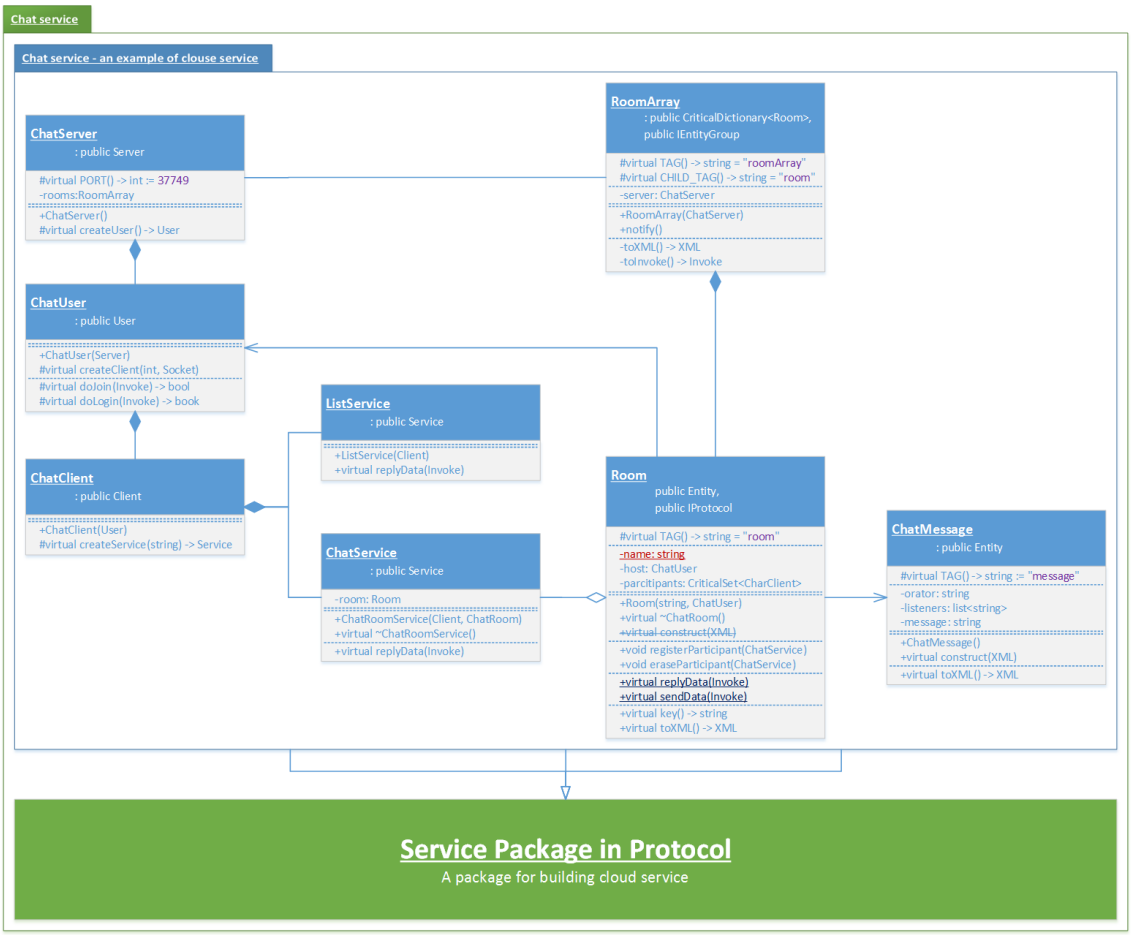
- ChatServer * server

  A chat server that rooms are belonged to.

6.9.1 Detailed Description

A manager of chat rooms.

An entity list of chat rooms and a manager of chat rooms taking a responsibility of notifying changes of children chat rooms to a related chat server.



Author

Jeongho Nam

6.9.2 Member Function Documentation

6.9.2.1 virtual auto CHILD_TAG ( ) const -> std::string [inline], [protected], [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

6.9.2.2 void notify ( )

Notify changes of chat rooms.

Notify changes of chat rooms to all clients in the chat server.

The documentation for this class was generated from the following files:

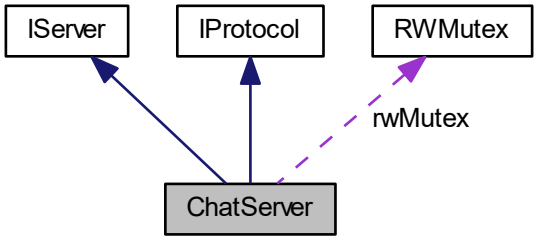
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatRoomArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatRoomArray.cpp

6.10 ChatServer Class Reference

A chat server.

`#include <ChatServer.hpp>`

Collaboration diagram for ChatServer:



Public Member Functions

- ChatServer ()

  Default Constructor.
- void eraseClient (ChatClient *client)

  Erase client when disconnected.
- virtual void replyData (std::shared_ptr< protocol::Invoke >) override

  Handling replied message.
- virtual void sendData (std::shared_ptr< protocol::Invoke >) override

  Sending a message.

Protected Member Functions

- virtual auto PORT () const -> int override

  Port number: #33749.
- virtual void addClient (protocol::Socket *socket) override

  Add a client.

Private Attributes

- std::set< ChatClient * > clientSet

  List of clients.
- library::RWMutex rwMutex

  A rw_mutex managing clientSet.

Additional Inherited Members

6.10.1 Detailed Description

A chat server.

Author

Jeongho Nam

6.10.2 Member Function Documentation

6.10.2.1 void replyData (std::shared_ptr< protocol::Invoke >) [override], [virtual]

Handling replied message.

When got a message from a client, sends the message to all clients.

Parameters

invoke	An invoke message represents a chat.

6.10.2.2 void sendData (std::shared_ptr< protocol::Invoke >) [override], [virtual]

Sending a message.

Send a message to all clients. Called by replyData().

Parameters

invoke	An invoke message represents a chat.

The documentation for this class was generated from the following files:

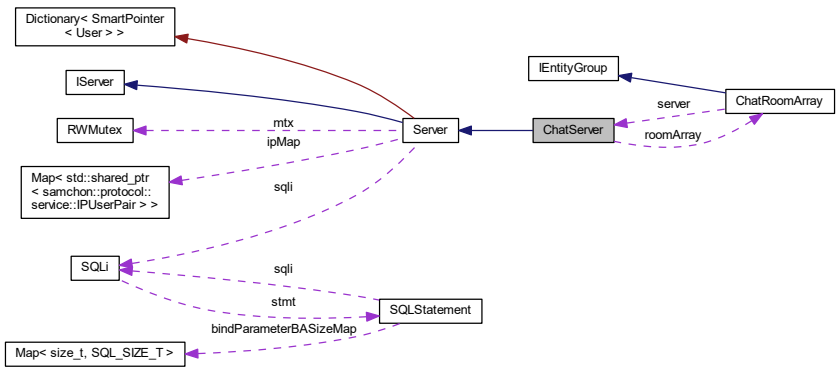
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/console_chat_server/ChatServer.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/console_chat_server/ChatServer.cpp

6.11 ChatServer Class Reference

A chat server.

`#include <ChatServer.hpp>`

Collaboration diagram for ChatServer:



Public Member Functions

- ChatServer ()

Default Constructor.

- auto getChatRoomArray () const -> ChatRoomArray *

  Get chat room list.

Protected Member Functions

- virtual auto PORT () const -> int override

  Port number of the server.

- virtual auto createUser () -> protocol::service::User *override

  Factory method of User.

Private Attributes

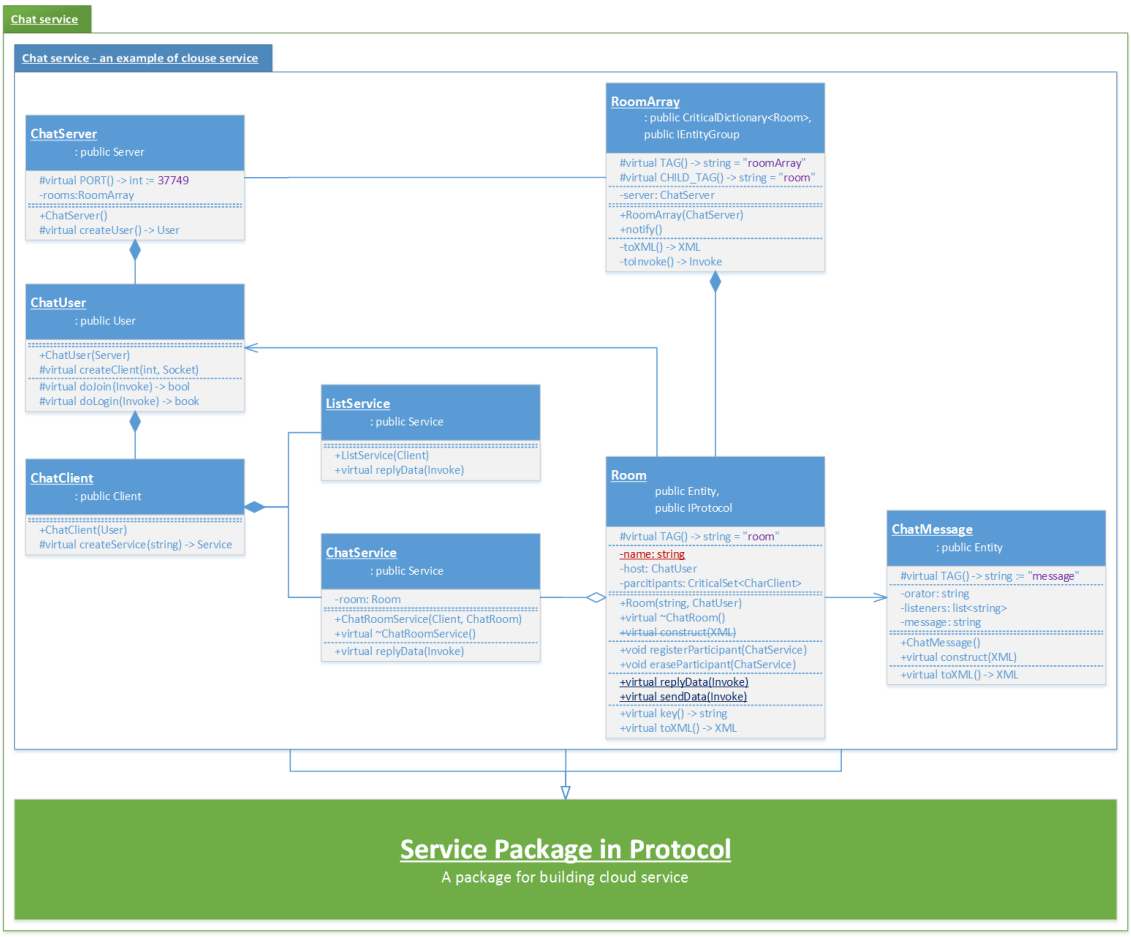
- ChatRoomArray * roomArray

  A manager of chat rooms.

Additional Inherited Members

6.11.1 Detailed Description

A chat server.



Author

Jeongho Nam

The documentation for this class was generated from the following file:

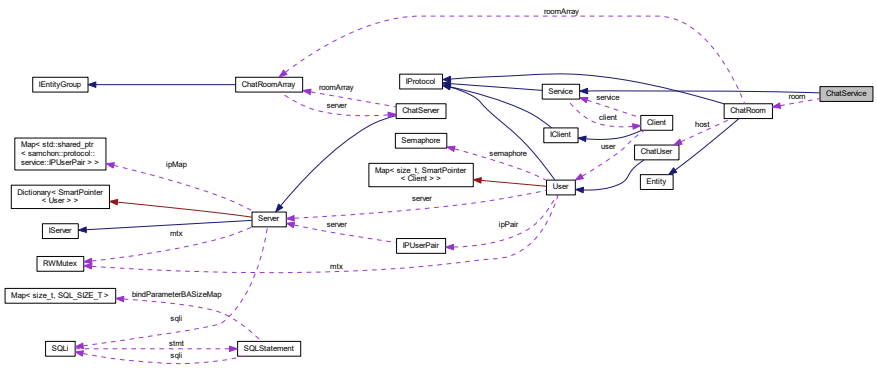
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatServer.hpp

6.12 ChatService Class Reference

A chat service.

`#include <ChatService.hpp>`

Collaboration diagram for ChatService:



Public Member Functions

- ChatService (protocol::service::Client ∗)

  Construct from a client.

Private Attributes

- ChatRoom ∗ room

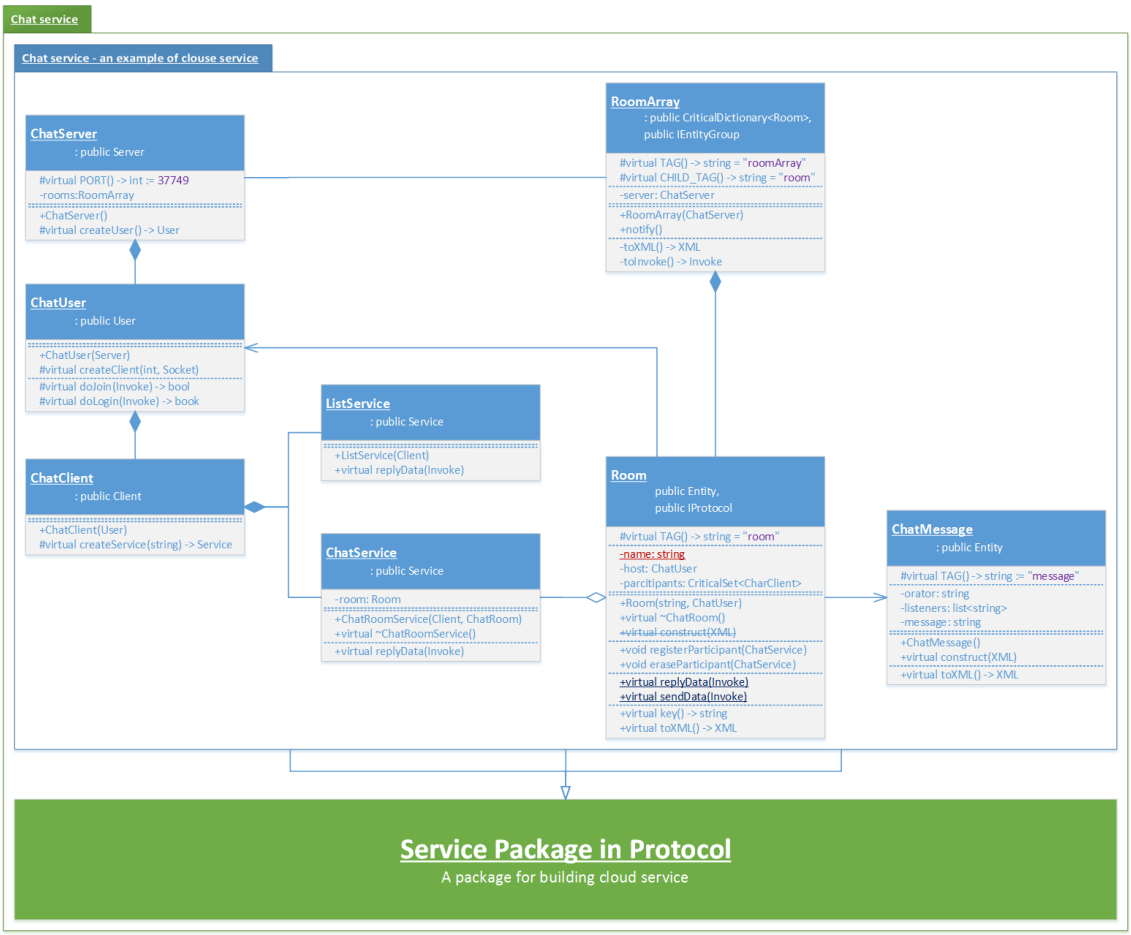
  A chat room.

Additional Inherited Members

6.12.1 Detailed Description

A chat service.

A chat service about a chat room.



Author

Jeongho Nam

6.12.2 Member Data Documentation

6.12.2.1 ChatRoom* room [private]

A chat room.

A chat room that related client is participating in.

The documentation for this class was generated from the following files:

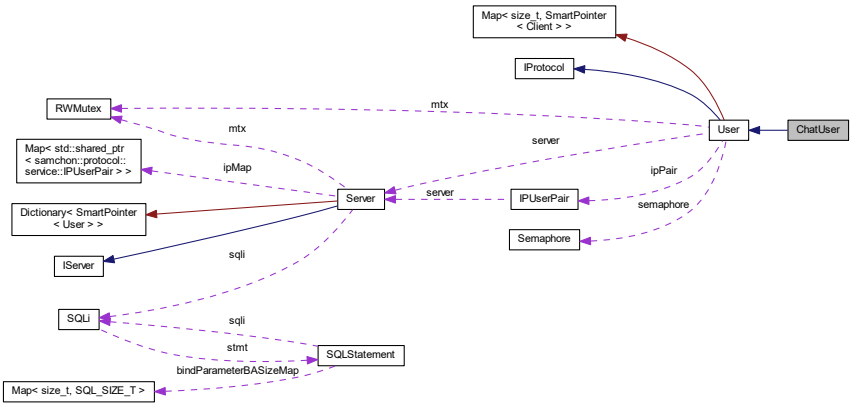
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatService.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatService.cpp

6.13 ChatUser Class Reference

An user in chat server.

```
#include <ChatUser.hpp>
```

Collaboration diagram for ChatUser:



Public Member Functions

- ChatUser (protocol::service::Server ∗)

  Construct from server.

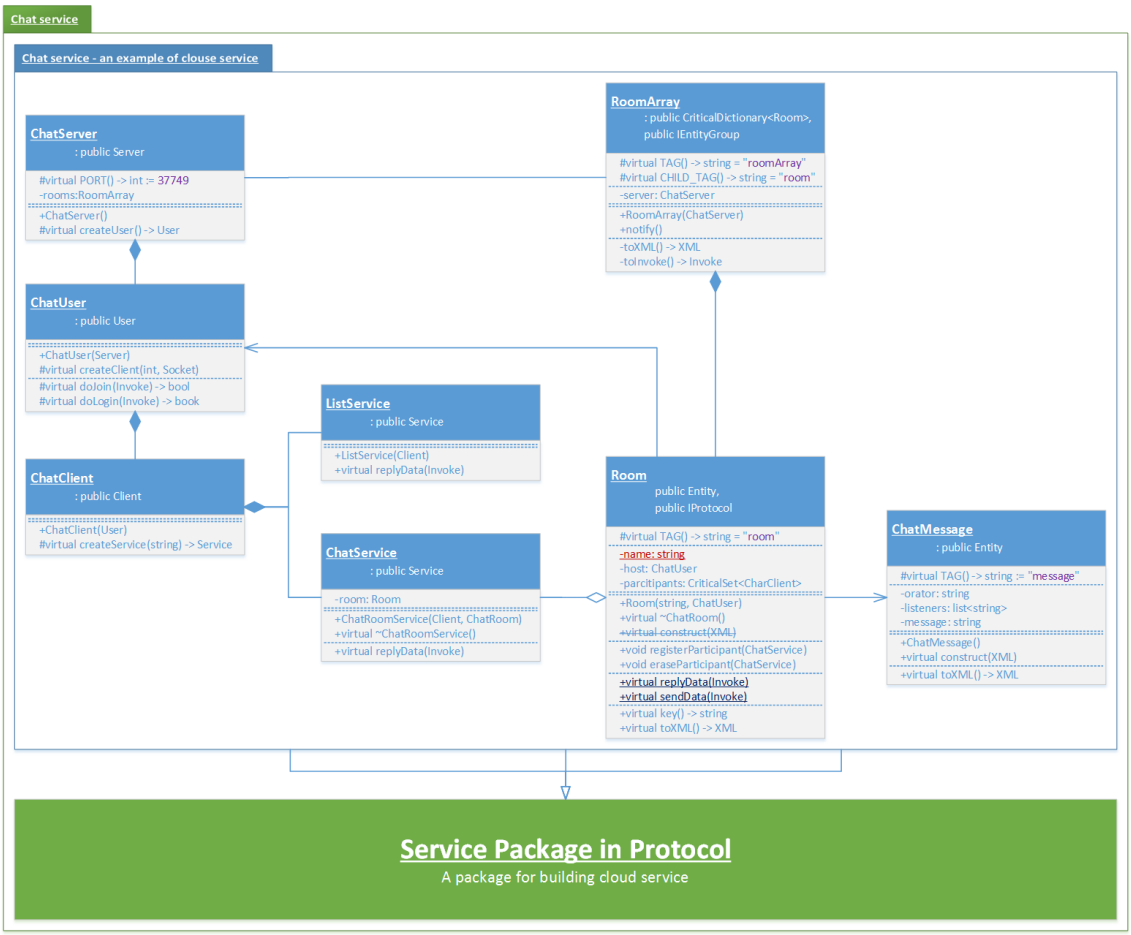
Protected Member Functions

- virtual auto createClient () -> protocol::service::Client ∗override

  Factory method of a Client.

6.13.1 Detailed Description

An user in chat server.



Author

Jeongho Nam

The documentation for this class was generated from the following file:

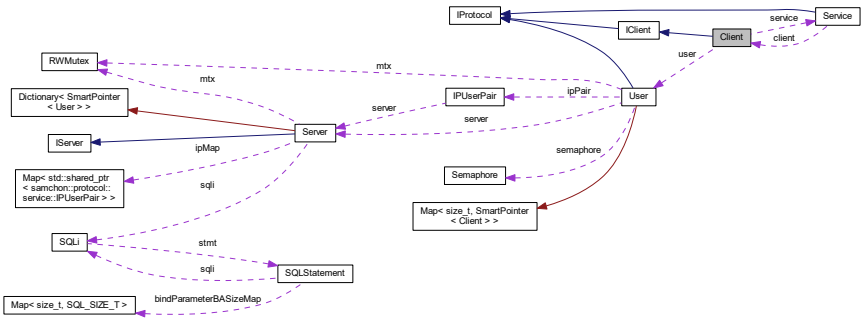
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ChatUser.hpp

6.14 Client Class Reference

A network boundary with the client in an User.

```
#include <Client.hpp>
```

Collaboration diagram for Client:



Public Member Functions

- Client (User ∗)

  Construct from User, No and socket.
- auto getUser () const -> User ∗

  Get User.
- auto getService () const -> Service ∗

  Get Service.
- auto getNo () const -> size_t

  Get no.
- void sendData (std::shared_ptr< Invoke >)

  Send Invoke message to (physical) client.
- void replyData (std::shared_ptr< Invoke >)

  Reply Invoke message from (physical) client.

Protected Member Functions

- virtual auto createService (const std::string &) -> Service ∗=0

  Factory method of Service.

Protected Attributes

- User ∗ user

  An User of the Client.
- size_t no

  A sequence number of the Client in an User.
- Service ∗ service

  A Service belongs to the Client.

Private Member Functions

- void constructService (const std::string &)

  Construct Service.

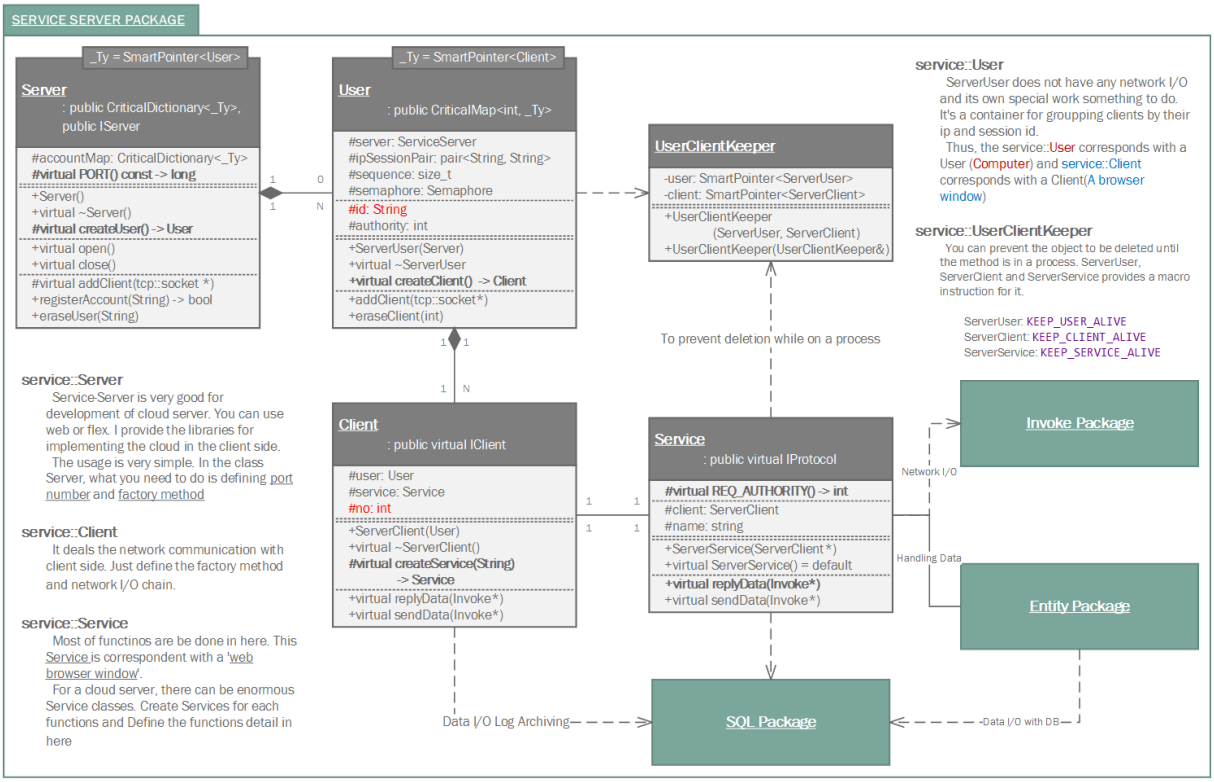
6.14.1 Detailed Description

A network boundary with the client in an User.

Client is an object interacting with physical client by socket.

The Client is correspond with a Window in JS (UI) 1:1

- A Window, the object in UI
- A Window, matching with window of a internet browser



Note

Method to override

- createUser()

See also

samchon::protocol
samchon::protocol::service

Author

Jeongho Nam

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Client (User * *user*)

Construct from User, No and socket.

Parameters

user	An User containing the Client

6.14.3 Member Function Documentation

6.14.3.1 virtual auto createService (const std::string &) -> Service * [protected], [pure virtual]

Factory method of Service.

Parameters

name	Requested name representing a Service from the (physical) client

Returns

A new Service belongs the the Client

Implemented in ChatClient.

6.14.3.2 void sendData (std::shared_ptr< Invoke >) [virtual]

Send Invoke message to (physical) client.

Method sendData of Client does not only send Invoke message to client system, but also archives the Invoke message to database system as a historical-log.

Note

Method sendData monopolies a critical section.

How many threads have called that sendData, the Invoke messages are delivered sequentially.

- Using sendData with a new thread is not recommended

Warning

You can't override this method, sendData.

Parameters

invoke	Invoke message to send

Reimplemented from IClient.

6.14.3.3 void replyData (std::shared_ptr< Invoke >) [virtual]

Reply Invoke message from (physical) client.

Handles replied Invoke message from client system.

1. Constructs Service
   - Constructs Service by requested service name.
   - Notifies client system whether the user satisfies the authority.
2. Shifts responsibility to related chain.

- Invoke message is about member (join, login, etc.), shift to User.
- Except that all, shifts to Service.

3. Archives the Invoke message to Database system as a historical-log.
   - Invoke to HISTORY_INVOKE table, with User's session and Client's sequence id.
   - InvokeParameters(s) to HISTORY_INVOKE_PARAMETER with key of HISTORY_INVOKE as foriegn key.

Note

Historical-log

HISTORY_INVOKE and HISTORY_INVOKE_PARAMETER has a 1:N relationship

If you want to modify archiving method, override method of Server::archiveReplyData()

Shifting responsibility to Service

Service::replyData will be called by a new thread. The thread will acquire an admission from semaphore in User to avoid exhausted allocation of threads.

Exception handling

If an std::exception has thrown from Service::replyData, the exception will archived in Database (HISTORY_INV↩OKE_EXCEPTION)

The most famous thrown parameters like std::exception, std::invalid_argument and std::runtime_error, those are all derived from std::exception.
-> http://www.cplusplus.com/reference/stdexcept/

- std::exception
  - std::logic_error
    - std::domain_error
    - std::invalid_argument
    - std::length_error
    - std::out_of_range
    - std::future_error
  - std::runtime_error
    - std::range_error
    - std::overflow_error
    - std::underflow_error
    - std::system_error

Warning

You can't override this method, replyData.

If you want to modify replyData's rule, override related objects' replyData

Parameters

invoke	Invoke message replied from client system

Reimplemented from IProtocol.

6.14.3.4 void constructService (const std::string & *name*) [private]

Construct Service.

Constructs Service by requested name from client and notify whether the User is satisfying authority.

Parameters

name	A name representing type of Service

The documentation for this class was generated from the following files:

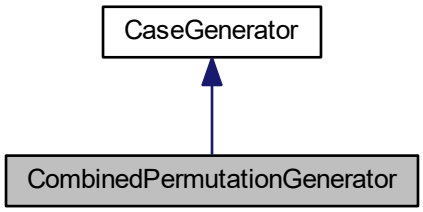
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/Client.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/Client.cpp

6.15 CombinedPermutationGenerator Class Reference

A combined-permutation case generator.

`#include <CombinedPermutationGenerator.hpp>`

Collaboration diagram for CombinedPermutationGenerator:



Public Member Functions

- CombinedPermutationGenerator (size_t n, size_t r)

  Construct from size of N and R.
- virtual auto operator[ ] (size_t) const -> std::vector< size_t > override

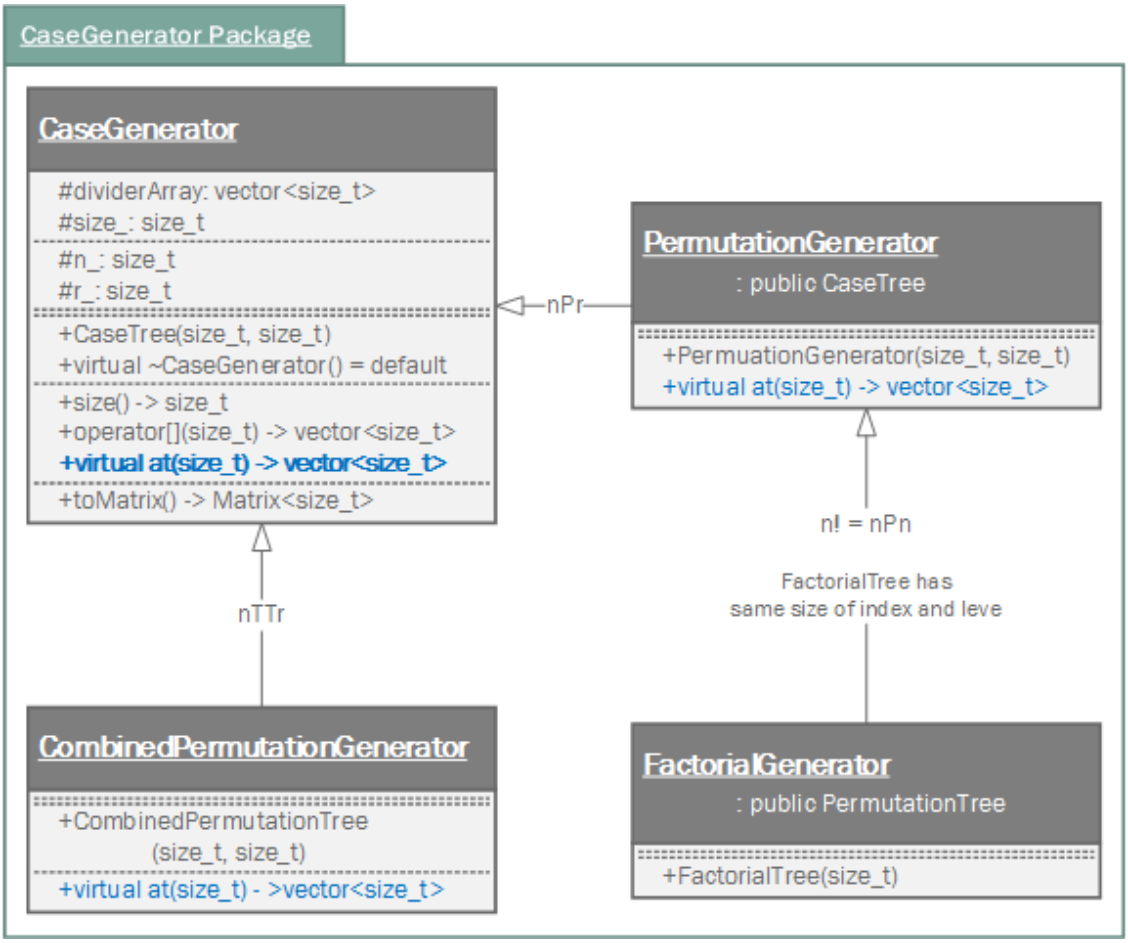
  Get x'th case.

Additional Inherited Members

6.15.1 Detailed Description

A combined-permutation case generator.

$_nTT_r$



See also

samchon::library

Author

Jeongho Nam

6.15.2 Constructor & Destructor Documentation

6.15.2.1 CombinedPermutationGenerator (size_t *n*, size_t *r*)

Construct from size of N and R.

Parameters

n	Size of candidates
r	Size of elements of each case

6.15.3 Member Function Documentation

6.15.3.1 auto operator[] (size_t) const -> std::vector< size_t > [override], [virtual]

Get x'th case.

Returns

The row of the x'th in combined permuation case

Implements CaseGenerator.

The documentation for this class was generated from the following files:

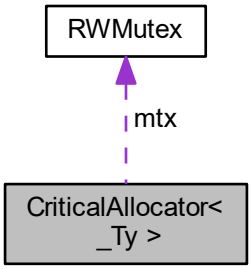
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/CombinedPermutationGenerator.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/CombinedPermutationGenerator.cpp

6.16 CriticalAllocator< _Ty > Class Template Reference

An allocator ensuring concurrency.

`#include <CriticalAllocator.hpp>`

Collaboration diagram for CriticalAllocator< _Ty >:



Public Member Functions

- auto getMutex () -> RWMutex &

  Get rw_mutex.

- template<class _U , class... _Args>
  void construct (_U *ptr, _Args &&...args)

  Lock when constructed.
  [Inherited] Construct an object.
  .

- template<class _U >
  void destroy (_U *ptr)

  Lock when destroyed
  [Inherited] Destory an object.

- auto allocate (size_type n, std::allocator< void >::const_pointer hint=NULL) -> pointer

  Lock when allocated
  [Inherited] Allocate block of storage.

- void deallocate (pointer ptr, size_type size)

  Lock when deallocated
  [Inherited] Release block of storage.

Private Attributes

- RWMutex mtx

  A rw_mutex for handling concurrency.

6.16.1 Detailed Description

template<class _Ty>class samchon::library::CriticalAllocator< _Ty >

An allocator ensuring concurrency.

CriticalAllocator is a std::allocator ensuring concurrency.

CriticalAllocator keeps safety in multi-threading environment automatically. But it ensures only safety. If a logic needs a mutual exclusion not only level of the container, it'd better to avoid using the CriticalAllocator and use RWMutex by yourself.

[Inherited]]

Allocators are classes that define memory models to be used by some parts of the Standard Library, and most specifically, by STL containers.

This section describes the default allocator template allocator (lowercase). This is the allocator that all standard containers will use if their last (and optional) template parameter is not specified, and is the only predefined allocator in the standard library. </p<

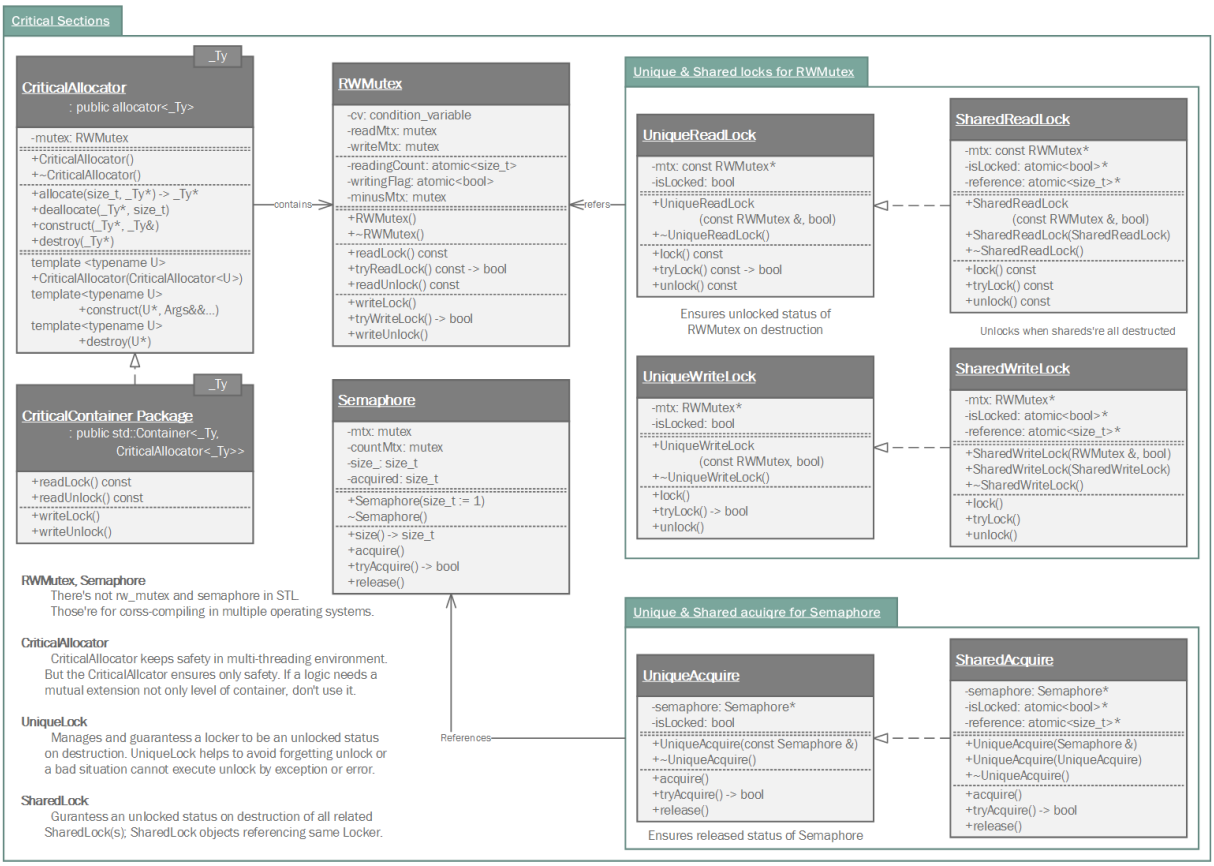
Other allocators may be defined. Any class Alloc for which allocator_traits<Alloc> produces a valid instantiation with the appropriate members defined can be used as an allocator on standard containers (Alloc may or may not implement the functionality through member functions). </p<

Except for its destructor, no member of the standard default allocator class template shall introduce data races. Calls to member functions that allocate or deallocate storage shall occur in a single total order, and each such deallocation shall happen before the next allocation (if any) in this order.

Technically, a memory model described by allocators might be specialized for each type of object to be allocated and even may store local data for each container they work with. Although this does not happen with the default allocator.

Referenced comments of std::allocator

- http://www.cplusplus.com/reference/memory/allocator/



Template Parameters

_Ty	Type of the elements allocated by the object (aliased as member type value_type).

See also

samchon::library

Author

Jeongho Nam

6.16.2 Member Function Documentation

6.16.2.1 void construct (_U * *ptr*, _Args &&... *args*) [inline]

Lock when constructed.
[Inherited] Construct an object.
.

Locks a mutex for concurrency when an elmented is constructed.

[Inherited]

Constructs an element object on the location pointed by p.

Notice that this does not allocate space for the element. It should already be available at p (see member allocate to allocate space).

Template Parameters

_U	
_Args	

Parameters

ptr	Pointer to a location with enough storage space to contain an element of type _U.
args	Arguments forwarded to the constructor. Args is a list of zero or more types.

6.16.2.2 void destroy (_U * *ptr*) [inline]

Lock when destroyed
[Inherited] Destory an object.

Locks a mutex for concurrency when an child is destroyed

[Inherited]

Destroys in-place the object pointed by p.

Notice that this does not deallocate the storage for the element (see member deallocate to release storage space).

Template Parameters

_U	

Parameters

ptr	Pointer to the object to be destroyed.

6.16.2.3 auto allocate (size_type *n*, std::allocator< void >::const_pointer *hint* = NULL) -> pointer [inline]

Lock when allocated
[Inherited] Allocate block of storage.

Locks a mutex for concurrency when children elements are deallocated.

[Inherited]

Attempts to allocate a block of storage with a size large enough to contain n elements of member type value_type (an alias of the allocator's template parameter), and returns a pointer to the first element.

The storage is aligned appropriately for objects of type value_type, but they are not constructed.

In the standard default allocator, the block of storage is allocated using ::operator new one or more times, and throws bad_alloc if it cannot allocate the total amount of storage requested.

Parameters

n	Number of elements (each of size sizeof(value_type)) to be allocated. The member type size_type is an alias of size_t (in the standard default allocator) size_t is an unsigned integral type.
hint	Either 0 or a value previously obtained by another call to allocate and not yet freed with deallocate. When it is not 0, this value may be used as a hint to improve performance by allocating the new block near the one specified. The address of an adjacent element is often a good choice.

Returns

A pointer to the initial element in the block of storage.

pointer and const_pointer are member types (defined as aliases of T∗ and const T∗ respectively in std::allocator).

The standard default allocator throws bad_alloc if it cannot allocate the requested amount of storage.

6.16.2.4 void deallocate (pointer *ptr*, size_type *size*) [inline]

Lock when deallocated
[Inherited] Release block of storage.

Locks a mutex for concurrency when children elements are deallocated.

[Inherited]

Releases a block of storage previously allocated with member allocate and not yet released.

The elements in the array are not destroyed by a call to this member function.

In the default allocator, the block of storage is at some point deallocated using ::operator delete (either during the function call, or later).

Parameters

ptr	Pointer to a block of storage previously allocated with allocator::allocate. pointer is a member type (defined as an alias of T∗ in std::allocator<_Ty>).
size	Number of elements allocated on the call to allocator::allocate for this block of storage. The member type size_type is an alias of size_t (in the standard default allocator). size_t is an unsigned integral type.

The documentation for this class was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/CriticalAllocator.hpp

6.17 Date Class Reference

Date (year, month, day in month)

`#include <Date.hpp>`

Public Member Functions

- Date ()

  Default Constructor.
- Date (const super &)

  Copy Constructor.
- Date (super &&)
- Date (int year, int month, int date)

  Construct from date elements.
- Date (const WeakString &)

  Constructor by std::string.
- Date (long long linuxTime)

  Construct from linux_time.
- void set (const std::string &)

  Setter by string.
- virtual void set (const WeakString &)

  Setter by weak_string.
- void set (int year, int month, int date)

  Setter by elements of date.
- void set (long long linuxTime)

  Setter by linux_time.
- virtual void setYear (int)

  Set year of the Date.
- virtual void setMonth (int)

  Set month of the Date.
- virtual void setDate (int)
- virtual void addYear (int)

  Add years to the Date.
- virtual void addMonth (int)

  Add months to the Date.
- virtual void addWeek (int)

  Add weeks to the Date.
- virtual void addDate (int)

  Add days to the Date.
- auto getYear () const -> int

  Get year of the Date.
- auto getMonth () const -> int

  Get month of the Date.
- auto getDate () const -> int

  Get the day in month of the Date.
- auto getDay () const -> int

  Get the day in week of the Date.
- auto toLinuxTime () const -> long long

  Converts the Date to linux_time.
- virtual auto toString () const -> std::string

  Converts the Date to std::string.

Protected Member Functions

- auto toTM () const -> struct::tm

  Converts the Date to struct tm.

Static Protected Member Functions

- static auto calcSeconds (int year, int month, int date) -> long long

  Calculates how many seconds have flowen since 0000-01-01 00:00:00.
- static auto calcLastDates (int year) -> std::array< int, 12 >

  Calculates and gets an array of final date of each month for that year.

Static Protected Attributes

- static std::chrono::system_clock::time_point TP_1970 = chrono::system_clock::from_time_t(0)

  time_point for 1970-01-01 09:00:00

6.17.1 Detailed Description

Date (year, month, day in month)

Date class is a chrono::time_point<std::chrono::system_clock>.

Date represents the date with year, month and day in month. The basic date (local time) is from your operating system.

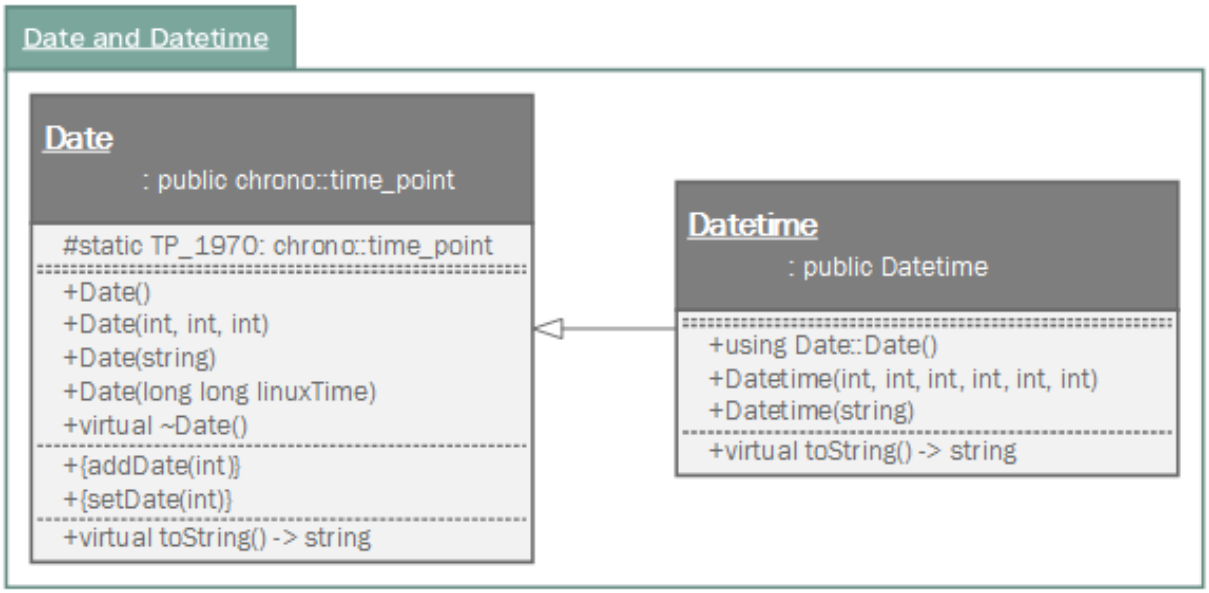
[Inherited]

A time_point object expresses a point in time relative to a clock's epoch.

Internally, the object stores an object of a duration type, and uses the Clock type as a reference for its epoch.

Referenced comments of std::chrono::time_point

- http://www.cplusplus.com/reference/chrono/time_point/



See also

samchon::library

Author

Jeongho Nam

6.17.2 Constructor & Destructor Documentation

6.17.2.1 Date ( )

Default Constructor.

The date will be now (Now's date from operating system)

6.17.2.2 Date (const super & *date*)

Copy Constructor.

Copy the date from another Date

Parameters

date	Target to be copied

6.17.2.3 Date (super && *date*)

Move Constructor

Gets the date from other and truncate the other

Parameters

date	Target to be moved

6.17.2.4 Date (int *year*, int *month*, int *date*)

Construct from date elements.

Parameters

year	ex) 2015
month	January to December. 1 $\sim$ 12
date	day in month. 1 $\sim$ 31

Exceptions

invalid_argument	month or date is invalid.

6.17.2.5 Date (const WeakString & *wStr*)

Constructor by std::string.

Parameters

str	A weak_string expressing the date. (1991-01-01)

Exceptions

invalid_argument	month or date is invalid.

6.17.2.6 Date (long long *linuxTime*)

Construct from linux_time.

Parameters

linuxTime	linux_time to be converted

6.17.3 Member Function Documentation

6.17.3.1 long long calcSeconds (int *year*, int *month*, int *date*) -> long long [static], [protected]

Calculates how many seconds have flowen since 0000-01-01 00:00:00.

This static method is used for calculate the linux_time (since 1970-01-01 09:00:00)

Returns

linux_time of the Date

6.17.3.2 auto calcLastDates (int *year*) -> std::array<int, 12> [static], [protected]

Calculates and gets an array of final date of each month for that year.

Parameters

year	The target year

Returns

Array of fianl date of each month for that year

6.17.3.3 void set (const std::string &)

Setter by string.

Parameters

str	Y-m-d

Exceptions

invalid_argument	month or date is invalid.

6.17.3.4 void set (const WeakString & *wStr*) [virtual]

Setter by weak_string.

Parameters

wstr	Y-m-d

Exceptions

invalid_argument	month or date is invalid.

Reimplemented in Datetime.

6.17.3.5 void set (int *year*, int *month*, int *date*)

Setter by elements of date.

Parameters

year	ex) 2015
month	January to December. 1 ∼ 12
date	day in month. 1 ∼ 31

Exceptions

invalid_argument	month or date is invalid.

6.17.3.6 void set (long long *linuxTime*)

Setter by linux_time.

Parameters

linuxTime	linux_time to be converted

6.17.3.7 void setYear (int *year*) [virtual]

Set year of the Date.

If previous date is leaf month's expiration date and target year what you want is not leaf, the date will be 28

- 2000-02-29 -> setYear(2001) -> 2001-02-28
- 2001-02-28 -> setYear(2000) -> 2000-02-08

Parameters

val	Target year

Reimplemented in Datetime.

6.17.3.8 void setMonth (int *month*) [virtual]

Set month of the Date.

If the expiration date of the month will be shrinked, the date will be changed to the expiration date

- 2000-03-31 -> setMonth(4) -> 2000-04-30
- 2007-08-31 -> setMonth(9) -> 2007-09-30

Parameters

val	Target month (1 - 12)

Exceptions

invalid_argument	month is out of range

Reimplemented in Datetime.

6.17.3.9 void setDate (int *val*) [virtual]

Set date of the Date

Parameters

val	Target day in month

Exceptions

invalid_argument	When date is over expiration date in month

Reimplemented in Datetime.

6.17.3.10 void addYear (int *val*) [virtual]

Add years to the Date.

Parameters

val	Years to add

Reimplemented in Datetime.

6.17.3.11 void addMonth (int *val*) [virtual]

Add months to the Date.

Not a matter to val is over 12. If the month is over 12, then years will be added.

Parameters

val	Months to add

Reimplemented in Datetime.

6.17.3.12 void addWeek (int *val*) [virtual]

Add weeks to the Date.

Not a matter that val is too huge. If the adding weeks derives modification in month or year, then it will be.

Parameters

val	Weeks to add

Reimplemented in Datetime.

6.17.3.13 void addDate (int *val*) [virtual]

Add days to the Date.

Not a matter that val is too huge. If the adding dates derives modification in month or year, then it will be.

Parameters

val	Days to add

Reimplemented in Datetime.

6.17.3.14 int getYear ( ) const -> int

Get year of the Date.

Returns

Year

6.17.3.15 int getMonth ( ) const -> int

Get month of the Date.

Returns

Month as integer (1: January, 12: December)

6.17.3.16 int getDate ( ) const -> int

Get the day in month of the Date.

Returns

Day in month (1 to 31)

6.17.3.17 int getDay ( ) const -> int

Get the day in week of the Date.

Returns

Day in week to integer (0: Sunday, 1: Monday, ..., 6: Saturday)

6.17.3.18 auto toTM ( ) const -> struct ::tm [protected]

Converts the Date to struct tm.

Returns

struct tm representing the Date

6.17.3.19 auto toLinuxTime ( ) const -> long long

Converts the Date to linux_time.

Returns

linux_time of the Date

6.17.3.20 auto toString ( ) const -> std::string [virtual]

Converts the Date to std::string.

Returns

std::string expressing the Date

Reimplemented in Datetime.

6.17.4 Member Data Documentation

6.17.4.1 chrono::system_clock::time_point TP_1970 = chrono::system_clock::from_time_t(0) [static], [protected]

time_point for 1970-01-01 09:00:00

It's the zero(0) in linux_time

The documentation for this class was generated from the following files:

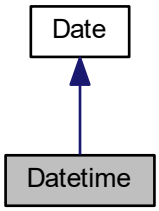
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Date.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Date.cpp

6.18 Datetime Class Reference

Date and time.

`#include <Datetime.hpp>`

Collaboration diagram for Datetime:



Public Member Functions

- Datetime ()
- Datetime (const super::super &)

  Copy Constructor.
- Datetime (super::super &&)
- Datetime (int year, int month, int date, int hour=0, int min=0, int sec=0)

  Construct from datetime elements.
- Datetime (const WeakString &)

Construct from WeakString.

- Datetime (long long linuxTime)

  Construct from linux_time.
- virtual void set (const WeakString &) override

  Set by weak_string.
- void set (int year, int month, int date, int hours=0, int minutes=0, int seconds=0)
- void set (long long linuxTime)
- virtual void setYear (int)

  Set year of the Date.
- virtual void setMonth (int)

  Set month of the Date.
- virtual void setDate (int)
- void setHour (int)

  Set hour.
- void setMinute (int)

  Set minute.
- void setSecond (int)

  Set hour.
- virtual void addYear (int) override

  Add years to the Date.
- virtual void addMonth (int) override

  Add months to the Date.
- virtual void addWeek (int) override

  Add weeks to the Date.
- virtual void addDate (int) override

  Add days to the Date.
- void addHour (int)

  Add hours.
- void addMinute (int)

  Add minutes.
- void addSecond (int)

  Add seconds.
- auto getHour () const -> int

  Get hour.
- auto getMinute () const -> int

  Get minute.
- auto getSecond () const -> int

  Get second.
- virtual auto toString () const -> std::string override

  Converts the Datetime to std::string.

Additional Inherited Members

6.18.1 Detailed Description

Date and time.

Datetime is a Date having hours, minutes and seconds addictionally.

[Inherited]

Date class is a chrono::time_point<std::chrono::system_clock>.

Date represents the date with year, month and day in month. The basic date (local time) is from your operating system.

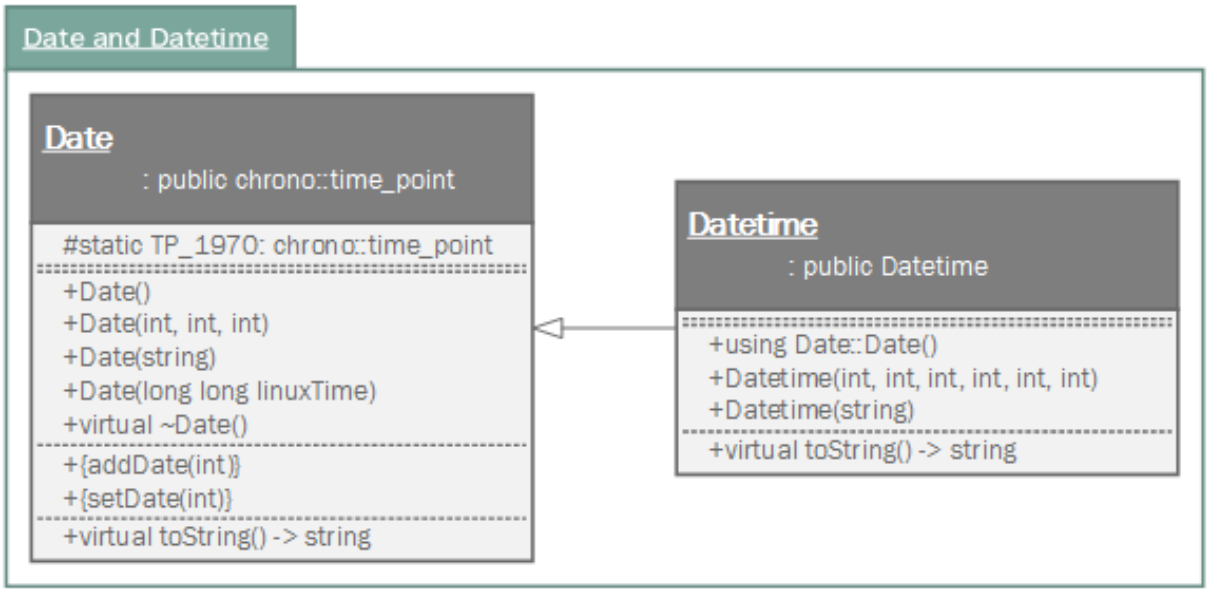
[Inherited]

A time_point object expresses a point in time relative to a clock's epoch.

Internally, the object stores an object of a duration type, and uses the Clock type as a reference for its epoch.

Referenced comments of std::chrono::time_point

- http://www.cplusplus.com/reference/chrono/time_point/



See also

samchon::library

Author

Jeongho Nam

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Datetime ( )

Default Constructor

The datetime will be now (Now from operating system)

6.18.2.2 Datetime (const super::super & *date*)

Copy Constructor.

Copy the date from another Date

Parameters

date	Target to be copied

6.18.2.3 Datetime (super::super && *date*)

Move Constructor

Gets the date from other and truncate the other

Parameters

date	Target to be moved

6.18.2.4 Datetime (int *year*, int *month*, int *date*, int *hour* = 0, int *min* = 0, int *sec* = 0)

Construct from datetime elements.

Parameters

year	ex) 2015
month	January to December. 1 $\sim$ 12
date	Day in month. 1 $\sim$ 31
hour	Hours (0-23)
min	Minutes (0-59)
sec	Seconds (0-59)

Exceptions

invalid_argument	parameter is out of its own range.

6.18.2.5 Datetime (const WeakString & *wStr*)

Construct from WeakString.

Parameters

wstr	A weak_string expressing the date. (1991-01-01 09:27:03)

Exceptions

invalid_argument	parameter is out of its own range.

6.18.2.6 Datetime (long long *linuxTime*)

Construct from linux_time.

Parameters

linuxTime	linux_time to be converted

6.18.3 Member Function Documentation

6.18.3.1 void set (const WeakString & *wStr*) [override], [virtual]

Set by weak_string.

Parameters

wstr	A weak_string expressing the date. (1991-01-01 09:27:03)

Exceptions

invalid_argument	parameter is out of its own range.

Reimplemented from Date.

6.18.3.2 void set (int *year*, int *month*, int *date*, int *hours* = 0, int *minutes* = 0, int *seconds* = 0)

Exceptions

invalid_argument	parameter is out of range (ex: month -> 13)

6.18.3.3 void set (long long *linuxTime*)

6.18.3.4 void setYear (int *val*) [virtual]

Set year of the Date.

If previous date is leaf month's expiration date and target year what you want is not leaf, the date will be 28

- 2000-02-29 -> setYear(2001) -> 2001-02-28
- 2001-02-28 -> setYear(2000) -> 2000-02-08

Parameters

val	Target year

Reimplemented from Date.

6.18.3.5 void setMonth (int *val*) [virtual]

Set month of the Date.

If the expiration date of the month will be shrinked, the date will be changed to the expiration date

- 2000-03-31 -> setMonth(4) -> 2000-04-30
- 2007-08-31 -> setMonth(9) -> 2007-09-30

Parameters

val	Target month (1 - 12)

Exceptions

invalid_argument	month is out of range

Reimplemented from Date.

6.18.3.6 void setDate (int *val*) [virtual]

Set date of the Date

Parameters

val	Target day in month

Exceptions

invalid_argument	When date is over expiration date in month

Reimplemented from Date.

6.18.3.7 void setHour (int *val*)

Set hour.

Parameters

val	Target hour (0 - 23)

Exceptions

invalid_argument	When hour is out of range

6.18.3.8 void setMinute (int *val*)

Set minute.

Parameters

val	Target minute (0 - 59)

Exceptions

invalid_argument	When minute is out of range

6.18.3.9 void setSecond (int *val*)

Set hour.

Parameters

val	Target second (0 - 59)

Exceptions

invalid_argument	When second is out of range

6.18.3.10 void addYear (int *val*) [override], [virtual]

Add years to the Date.

Parameters

val	Years to add

Reimplemented from Date.

6.18.3.11 void addMonth (int *val*) [override], [virtual]

Add months to the Date.

Not a matter to val is over 12. If the month is over 12, then years will be added.

Parameters

val	Months to add

Reimplemented from Date.

6.18.3.12 void addWeek (int *val*) [override], [virtual]

Add weeks to the Date.

Not a matter that val is too huge. If the adding weeks derives modification in month or year, then it will be.

Parameters

val	Weeks to add

Reimplemented from Date.

6.18.3.13 void addDate (int *val*) [override], [virtual]

Add days to the Date.

Not a matter that val is too huge. If the adding dates derives modification in month or year, then it will be.

Parameters

val	Days to add

Reimplemented from Date.

6.18.3.14 void addHour (int *val*)

Add hours.

It's same with operator+=(chrono::hours(val))

Do not worry about the out of range of the hour(val) Parent items like date, month and years will be adjusted automatically.

Parameters

val	hours to add

6.18.3.15 void addMinute (int *val*)

Add minutes.

It's same with operator+=(chrono::minutes(val))

Do not worry about the out of range of the minutes(val) Parent items like date, month and years will be adjusted automatically.

Parameters

val	minutes to add

6.18.3.16 void addSecond (int *val*)

Add seconds.

It's same with operator+=(chrono::seconds(val))

Do not worry about the out of range of the seconds(val) Parent items like date, month and years will be adjusted automatically.

Parameters

val	seconds to add

6.18.3.17 int getHour ( ) const -> int

Get hour.

Returns

hour of the Datetime to integer

6.18.3.18 int getMinute ( ) const -> int

Get minute.

Returns

minute of the Datetime to integer

6.18.3.19 int getSecond ( ) const -> int

Get second.

Returns

second of the Datetime to integer

6.18.3.20 auto toString ( ) const -> std::string [override], [virtual]

Converts the Datetime to std::string.

Returns

std::string expressing the Datetime

Reimplemented from Date.

The documentation for this class was generated from the following files:

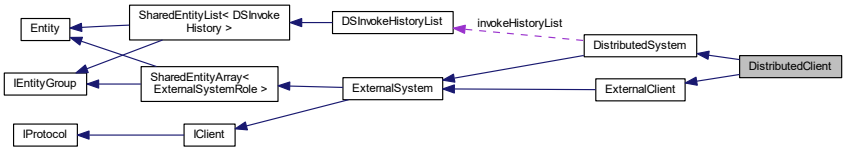
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Datetime.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Datetime.cpp

6.19 DistributedClient Class Reference

A network driver for a distributed client.

`#include <DistributedClient.hpp>`

Collaboration diagram for DistributedClient:



Public Member Functions

- DistributedClient ()

  Default Constructor.

Additional Inherited Members

6.19.1 Detailed Description

A network driver for a distributed client.

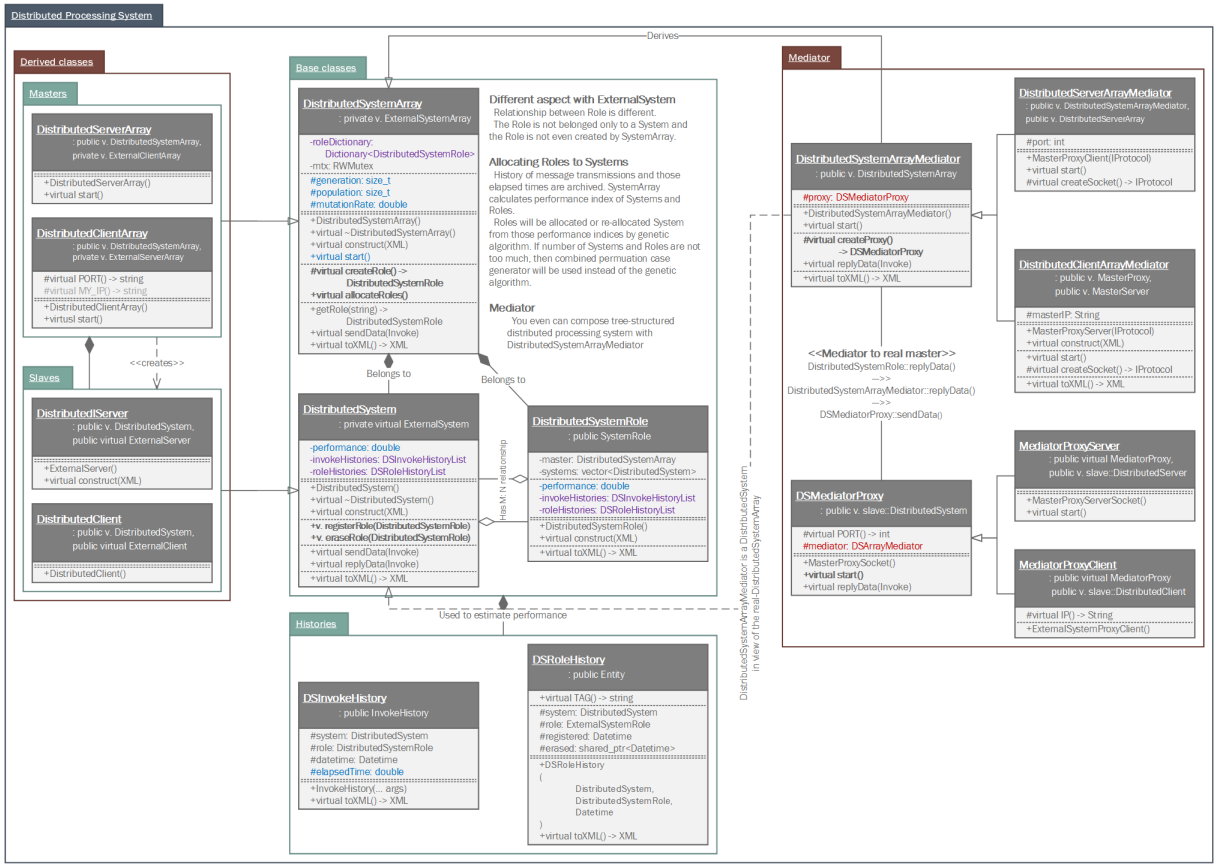
DistributedClient is an DistributedSystem specialized in client driver.

[Inherited]

DistributedSystem class is an ExternalSystem having role as DistributedSystemRole objects with performance index and history log allocation of DistributedSystemRole objects and have requsted Invoke message to related external system.

The greatest difference between DistributedSystem and ExternalSystem is, unlike the ExternalSystem class which having is ExternalSystemRole objects exclusively, DistributedSystem does not monopoly a DistributedSystemRole. The DistributedSystemRole can allocatedto multiple DistributedSystem.

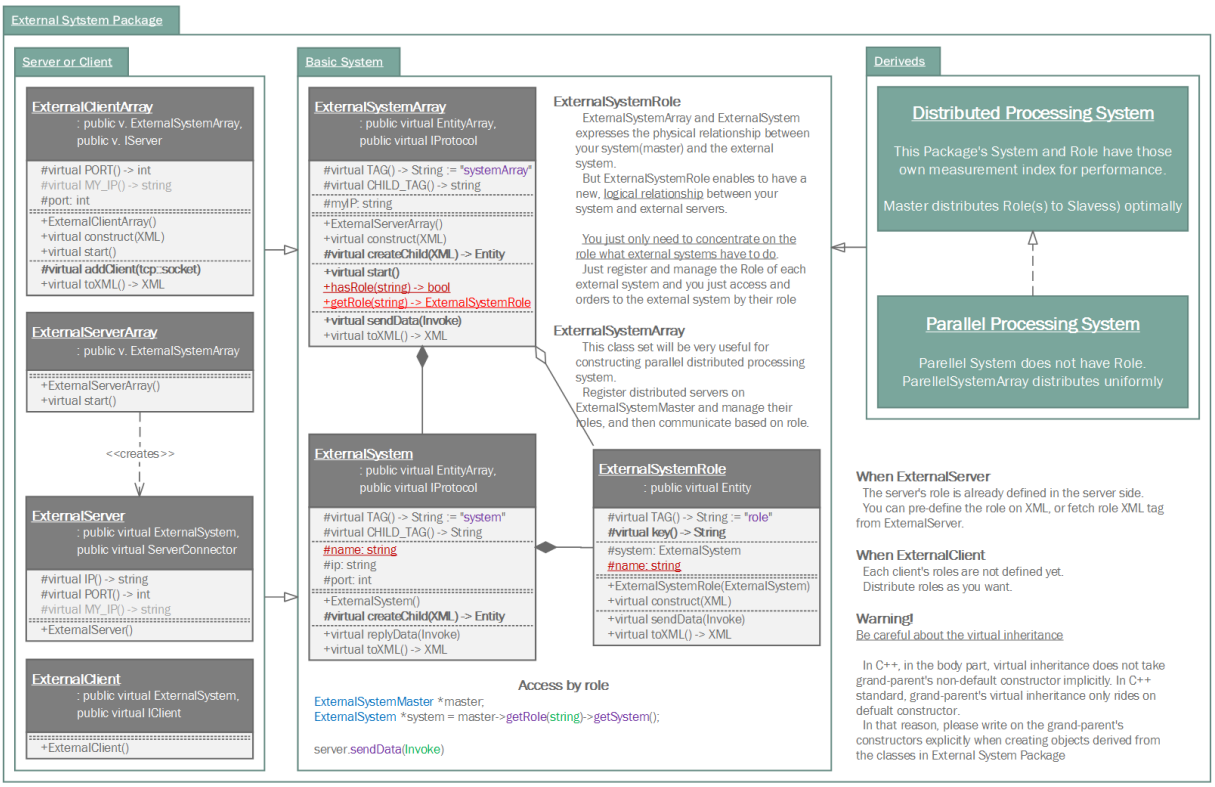
DistributedSystem and DistributedSystemRole classes have performance index and history log archiving what they've requested to the related distributed system. Allocating DistributedSystemRole objects to DistributedSyste will be determined those performance index and history log for optimize distribution of elasped time about each Invoke message processing.



[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, External↩System is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩ SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

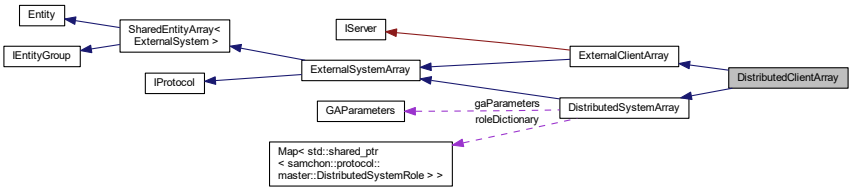
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedClient.hpp

6.20 DistributedClientArray Class Reference

An array of DistributedClient(s).

```
#include <DistributedClientArray.hpp>
```

Collaboration diagram for DistributedClientArray:



Public Member Functions

- DistributedClientArray ()

  Default Constructor.

- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.

- virtual void start () override

  Start interaction.

- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Additional Inherited Members

6.20.1 Detailed Description

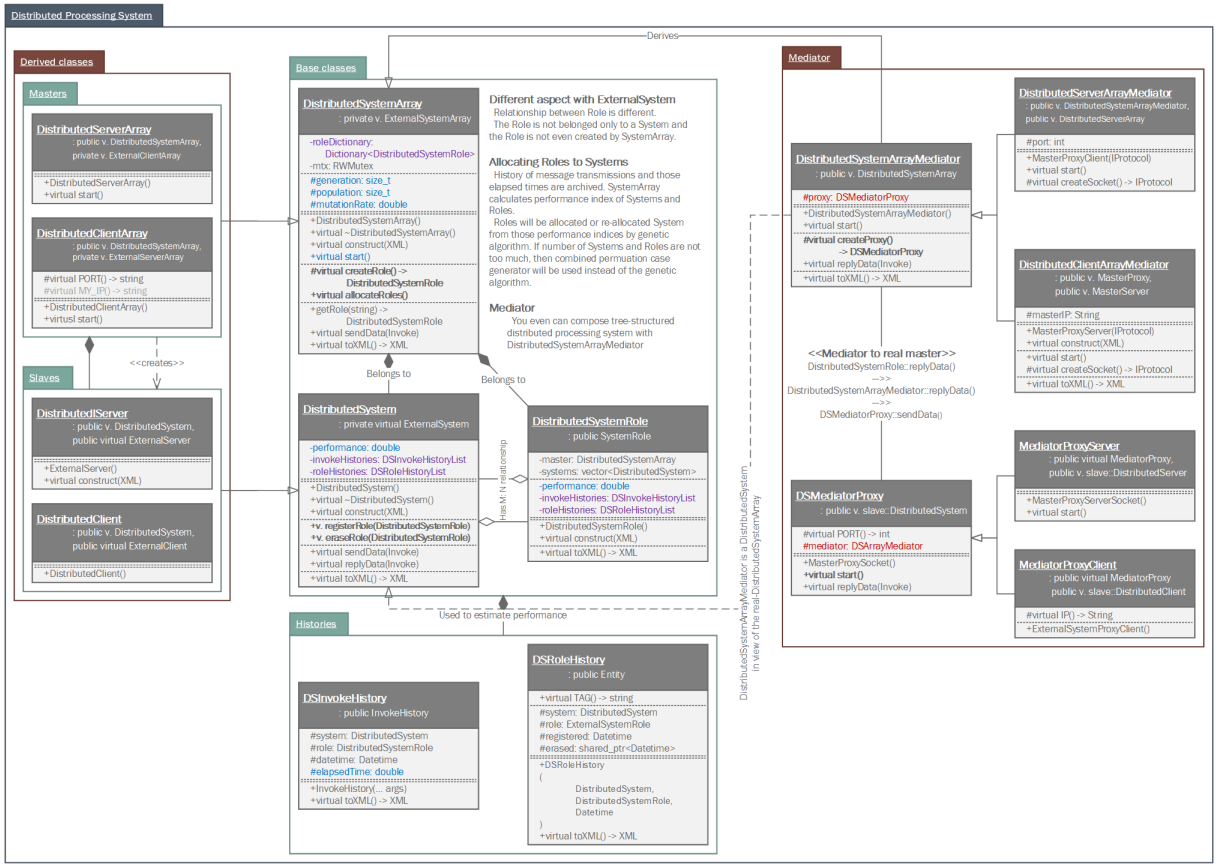
An array of DistributedClient(s).

DistributedClientArray is an DistributedSystemArray specialized in clients' driver.

[Inherited]

DistributedSystemArray class is an ExternalSystemArray and an abstract class containing and managing distributed system drivers within framework of master. The DistributedSystemArray class allocates DistributedSystemRole objects to DistributedSystem objects for optimization.

Unlike the ExternalSystemRole class which is dependent on an ExternalSystem, DistributedSystemRole is not dependent on a DistributedSystem and even can be allocated to multiple DistributedSystem objects. Allocation of DistributedSystemRole(s) to DistributedSystem(s) is determined by estimation of performance index pursuing best distribution. If number of DistributedSystem and DistributedSystemRole objects is not a lot, optimize by combined permuation algorithm. Else the objects are a lot, the allocation will be optimized by genetic algorithm.



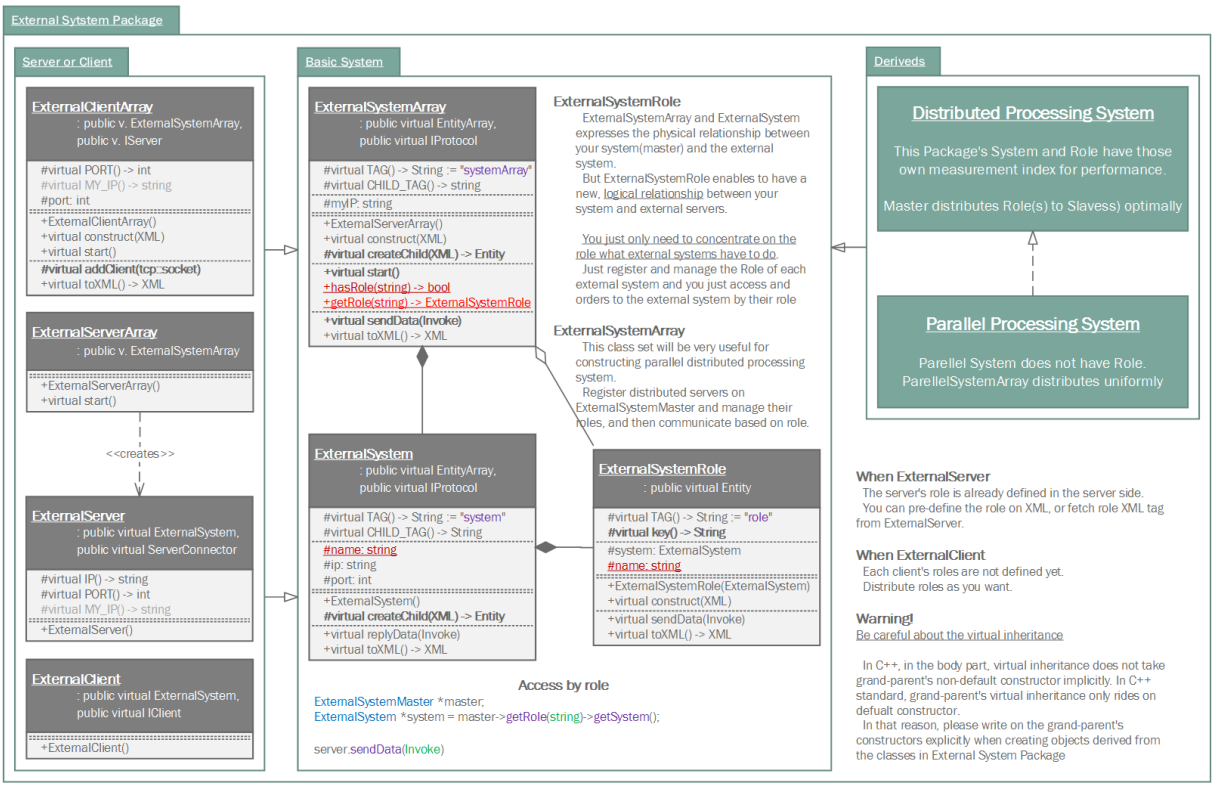
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.20.2 Member Function Documentation

6.20.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from ExternalClientArray.

Reimplemented in DistributedClientArrayMediator.

6.20.2.2 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Reimplemented from ExternalClientArray.

Reimplemented in DistributedClientArrayMediator.

6.20.2.3 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ExternalClientArray.

Reimplemented in DistributedClientArrayMediator.

The documentation for this class was generated from the following file:

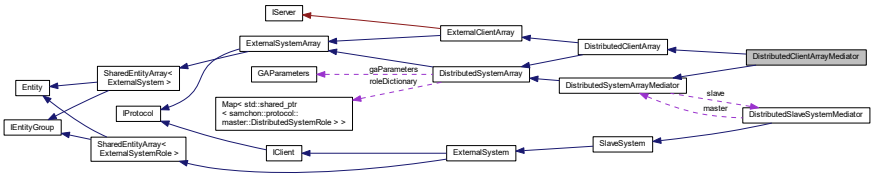
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedClientArray.hpp

6.21 DistributedClientArrayMediator Class Reference

A mediator of distributed system between master and slaves.

`#include <DistributedClientArrayMediator.hpp>`

Collaboration diagram for DistributedClientArrayMediator:



Public Member Functions

- DistributedClientArrayMediator ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual void start () override

  Start interaction.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Additional Inherited Members

6.21.1 Detailed Description

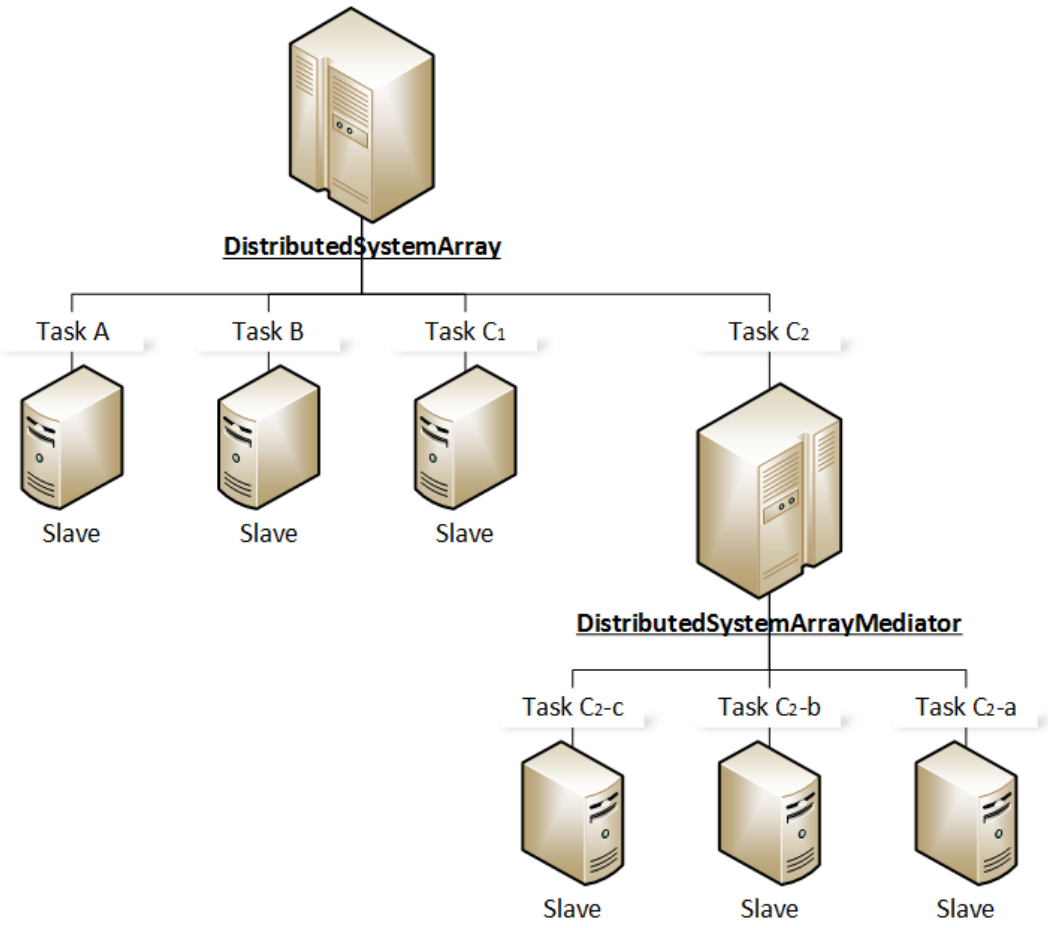
A mediator of distributed system between master and slaves.

DistributedClientArrayMediator is a DistributedSystemArrayMediator specialized in clients' driver. DistributedClientArrayMediator is a client for its master and also manages children clients' slaves.

[Inherited]

DistributedSystemArrayMediator class is used to realizing a mediator system of tree-structured distributed processing system. DistributedSystemArrayMediator intermediates between origin-master(DistributedSystemArray) and its slaves(slave::DistributedSystem).

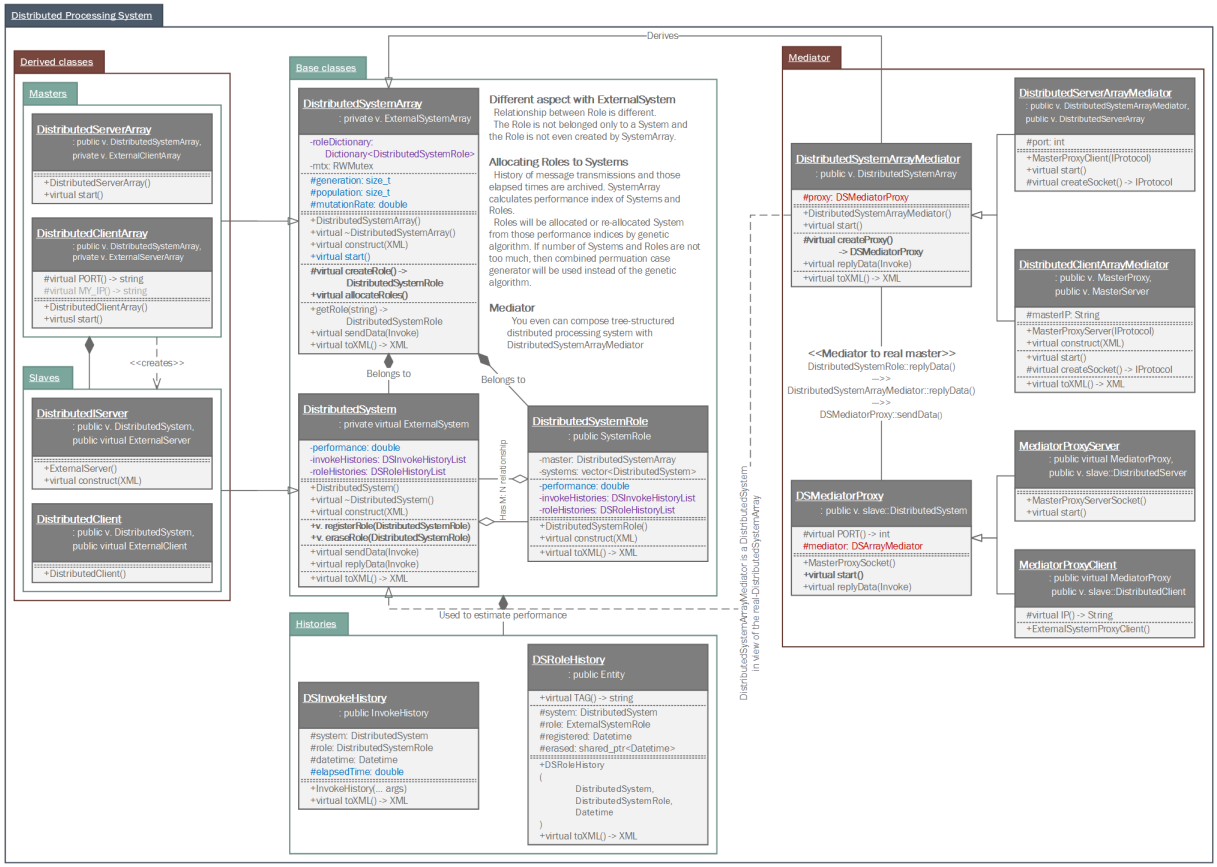
DistributedSystemArrayMediator is a master(DistributedSystemArray) within the framework of slaves systems (slave::DistributedSystem objects) and also can be a slave system (slave::DistributedSystem) within framework of its master(DistributedSystemArray) by having DistributedSlaveSystemMediator.



[Inherited]

DistributedSystemArray class is an ExternalSystemArray and an abstract class containing and managing distributed system drivers within framework of master. The DistributedSystemArray class allocates DistributedSystemRole objects to DistributedSystem objects for optimization.

Unlike the ExternalSystemRole class which is dependent on an ExternalSystem, DistributedSystemRole is not dependent on a DistributedSystem and even can be allocated to multiple DistributedSystem objects. Allocation of DistributedSystemRole(s) to DistributedSystem(s) is determined by estimation of performance index pursuing best distribution. If number of DistributedSystem and DistributedSystemRole objects is not a lot, optimize by combined permuation algorithm. Else the objects are a lot, the allocation will be optimized by genetic algorithm.



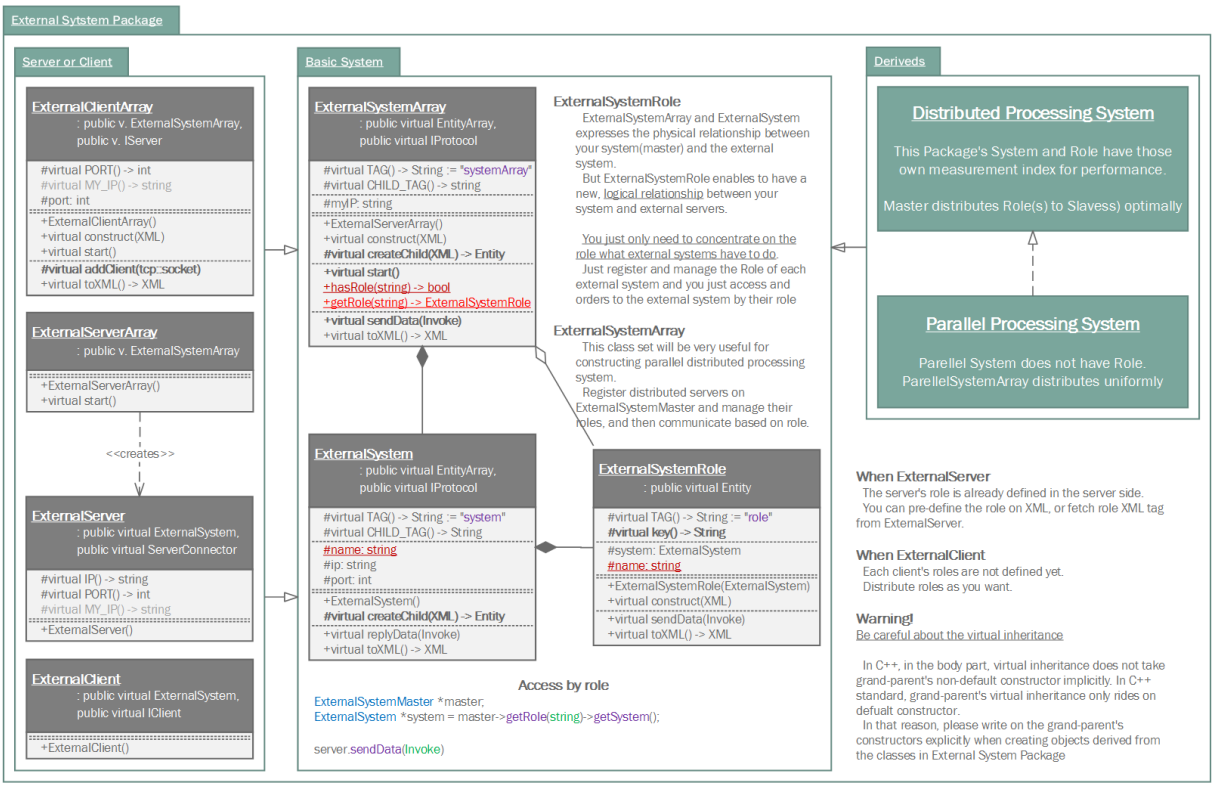
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.21.2 Member Function Documentation

6.21.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from DistributedClientArray.

6.21.2.2 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Reimplemented from DistributedClientArray.

6.21.2.3 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from DistributedClientArray.

The documentation for this class was generated from the following file:

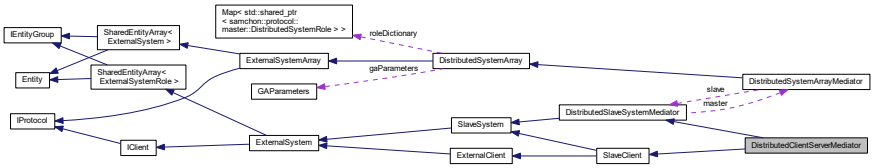
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedClientArrayMediator.hpp

6.22 DistributedClientServerMediator Class Reference

A distributed slave client for mediation.

`#include <DistributedSlaveClientMediator.hpp>`

Collaboration diagram for DistributedClientServerMediator:



Public Member Functions

- DistributedClientServerMediator ()

  Default Constructor.

Additional Inherited Members

6.22.1 Detailed Description

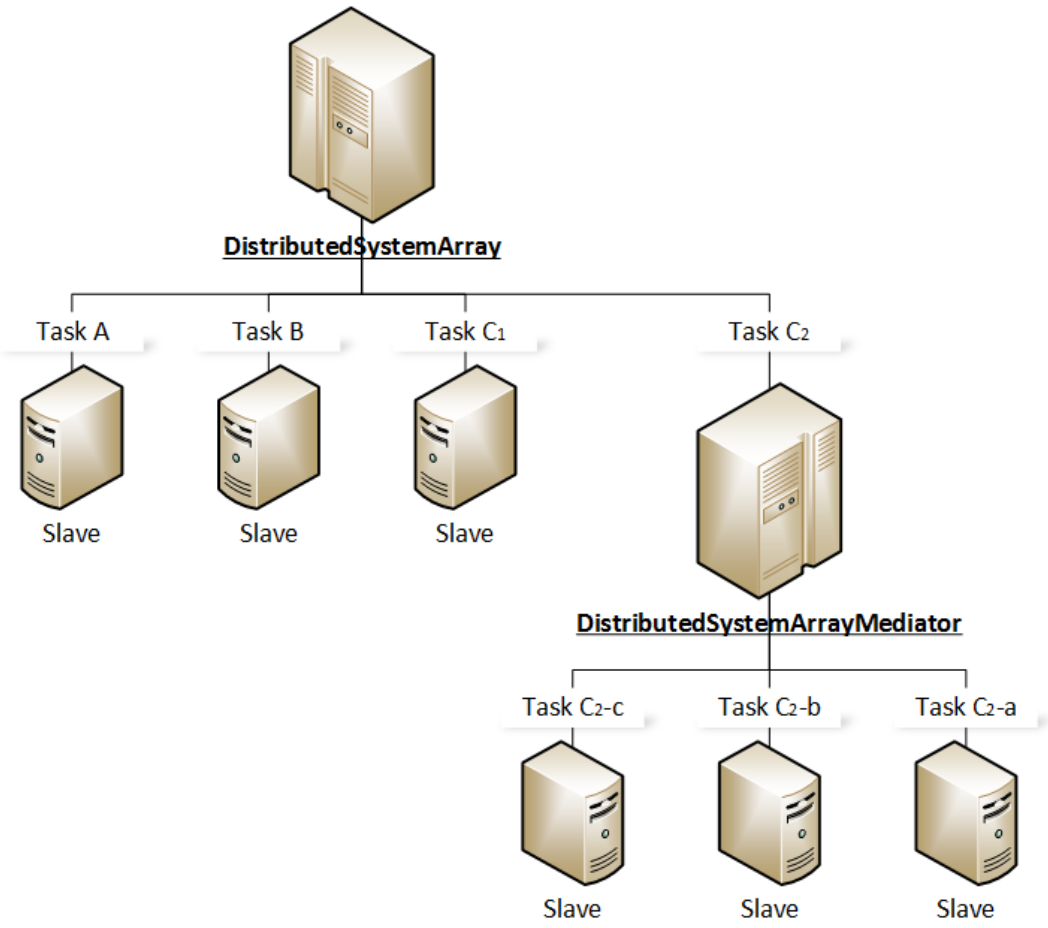
A distributed slave client for mediation.

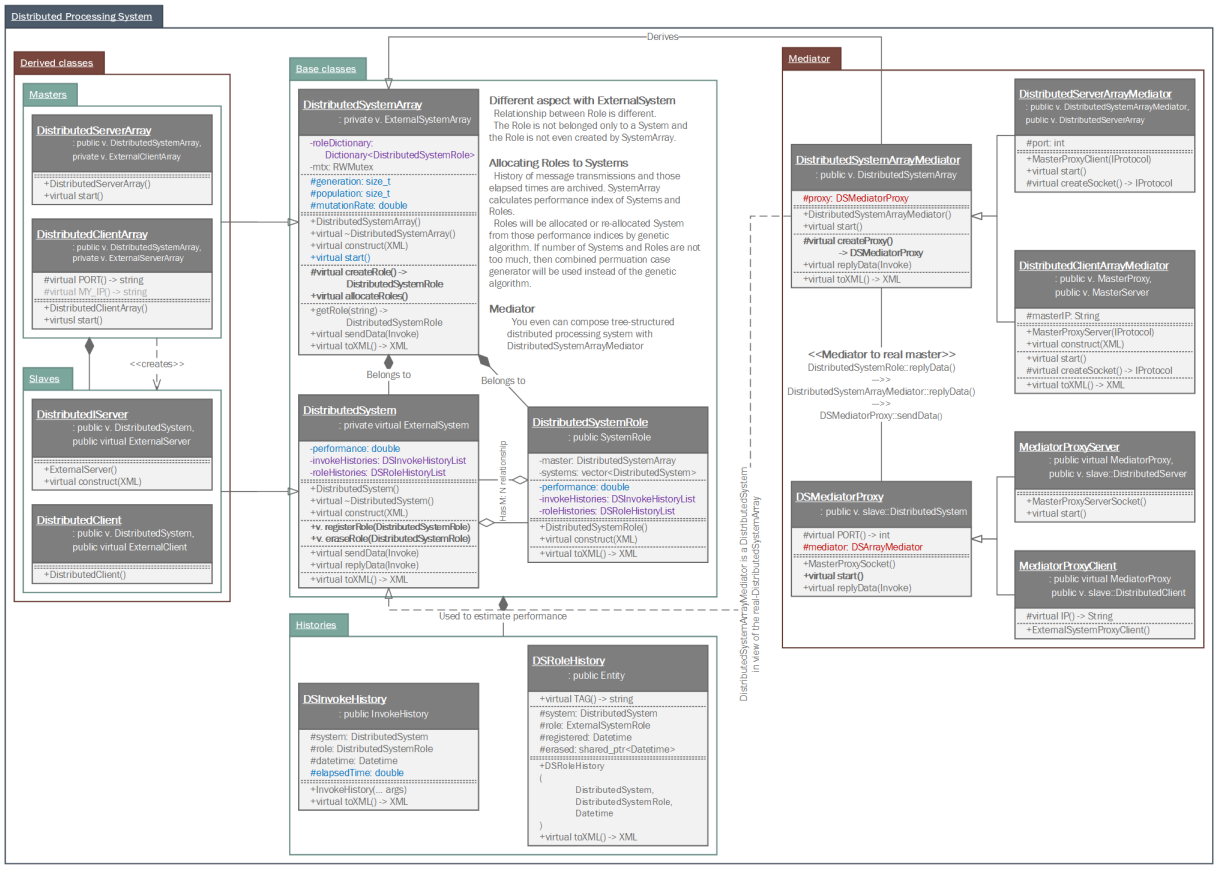
DistributedClientServerMediator is a DistributedSlaveSystemMediator class specialized in client driver.

[Inherited]

DistributedSlaveSystemMediator is an abstract class derived from slave::SlaveSystem for realizing tree-structured distriuted processins system. DistributedSlaveSystemMediator acts a slave system's role for its master, on the above level.

DistributedSystemArrayMediator and DistributedSlaveSystemMediator classes are paired to act a mediator system role by acting their own role, master system role for below slaves and slave system role for above master system.


DistributedSystemArray
Task A
Task B
Task C1
Task C2
Slave
Slave
Slave
DistributedSystemArrayMediator
Task C2-c
Task C2-b
Task C2-a
Slave
Slave
Slave


Distributed Processing System
Derived classes
Masters
DistributedServerArray
: public v. DistributedSystemArray,
private v. ExternalClientArray
+DistributedServerArray()
+virtual start()
DistributedClientArray
: public v. DistributedSystemArray,
private v. ExternalServerArray
#virtual PORT() -> string
#virtual MY_IP() -> string
+DistributedClientArray()
+virtual start()
<<creates>>
Slaves
DistributedIServer
: public v. DistributedSystem,
public virtual ExternalServer
+ExternalServer()
+virtual construct(XML)
DistributedClient
: public v. DistributedSystem,
public virtual ExternalClient
+DistributedClient()
Derives
Base classes
DistributedSystemArray
: private v. ExternalSystemArray
-roleDictionary:
Dictionary<DistributedSystemRole>
-mtx: RWMutex
#generation: size_t
#population: size_t
#mutationRate: double
+DistributedSystemArray()
+virtual ~DistributedSystemArray()
+virtual construct(XML)
+virtual start()
#virtual createRole() ->
DistributedSystemRole
+virtual allocateRoles()
+getRole(string) ->
DistributedSystemRole
+virtual sendData(Invoke)
+virtual toXML() -> XML
Different aspect with ExternalSystem
Relationship between Role is different.
The Role is not belonged only to a System and the Role is not even created by SystemArray.
Allocating Roles to Systems
History of message transmissions and those elapsed times are archived. SystemArray calculates performance index of Systems and Roles.
Roles will be allocated or re-allocated System from those performance indices by genetic algorithm. If number of Systems and Roles are not too much, then combined permuation case generator will be used instead of the genetic algorithm.
Mediator
You even can compose tree-structured distributed processing system with DistributedSystemArrayMediator
Belongs to
Belongs to
DistributedSystem
: private virtual ExternalSystem
-performance: double
-invokeHistories: DSInvokeHistoryList
-roleHistories: DSRoleHistoryList
+DistributedSystem()
+virtual ~DistributedSystem()
+virtual construct(XML)
+v. registerRole(DistributedSystemRole)
+v. eraseRole(DistributedSystemRole)
+virtual sendData(Invoke)
+virtual replyData(Invoke)
+virtual toXML() -> XML
Has N, N relationship
DistributedSystemRole
: public SystemRole
-master: DistributedSystemArray
-systems: vector<DistributedSystem>
-performance: double
-invokeHistories: DSInvokeHistoryList
-roleHistories: DSRoleHistoryList
+DistributedSystemRole()
+virtual construct(XML)
+virtual toXML() -> XML
Used to estimate performance
Histories
DSInvokeHistory
: public InvokeHistory
#system: DistributedSystem
#role: DistributedSystemRole
#datetime: Datetime
#elapsedTime: double
+InvokeHistory(... args)
+virtual toXML() -> XML
DSRoleHistory
: public Entity
+virtual TAG() -> string
#system: DistributedSystem
#role: ExternalSystemRole
#registered: Datetime
#erased: shared_ptr<Datetime>
+DSRoleHistory
(
DistributedSystem,
DistributedSystemRole,
Datetime
)
+virtual toXML() -> XML
DistributedSystemArrayMediator is a DistributedSystem
in view of the real-DistributedSystemArray
Mediator
DistributedSystemArrayMediator
: public v. DistributedSystemArray
#proxy: DSMediatorProxy
+DistributedSystemArrayMediator()
+virtual start()
#virtual createProxy()
-> DSMediatorProxy
+virtual replyData(Invoke)
+virtual toXML() -> XML
DistributedServerArrayMediator
: public v. DistributedSystemArrayMediator,
public v. DistributedServerArray
#port: int
+MasterProxyClient(IProtocol)
+virtual start()
#virtual createSocket() -> IProtocol
DistributedClientArrayMediator
: public v. MasterProxy,
public v. MasterServer
#masterIP: String
+MasterProxyServer(IProtocol)
+virtual construct(XML)
+virtual start()
#virtual createSocket() -> IProtocol
+virtual toXML() -> XML
<<Mediator to real master>>
DistributedSystemRole::replyData()
-->>
DistributedSystemArrayMediator::replyData()
-->>
DSMediatorProxy::sendData()
DSMediatorProxy
: public v. slave::DistributedSystem
#virtual PORT() -> int
#mediator: DSArrayMediator
+MasterProxySocket()
+virtual start()
+virtual replyData(Invoke)
MediatorProxyServer
: public virtual MediatorProxy,
public v. slave::DistributedServer
+MasterProxyServerSocket()
+virtual start()
MediatorProxyClient
: public virtual MediatorProxy
public v. slave::DistributedClient
#virtual IP() -> String
+ExternalSystemProxyClient()

[Inherited]

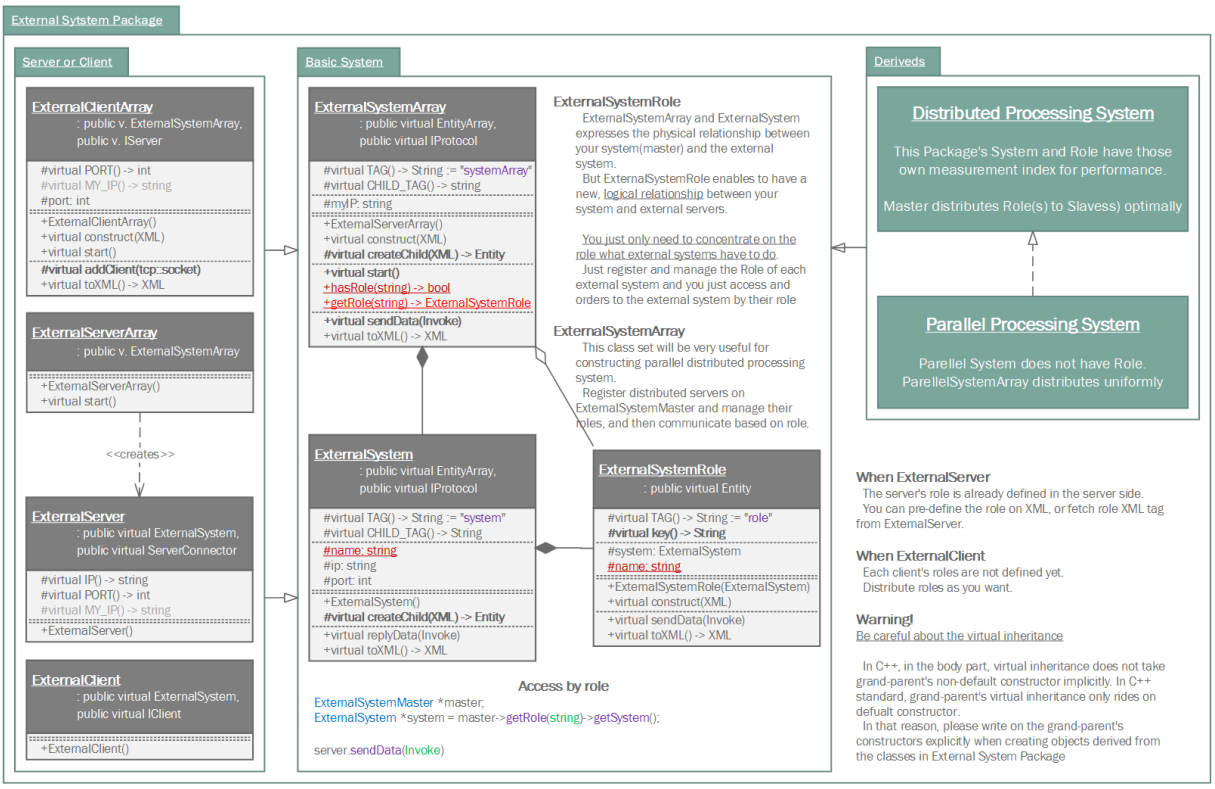
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::ParallelSystem and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, External↩System is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

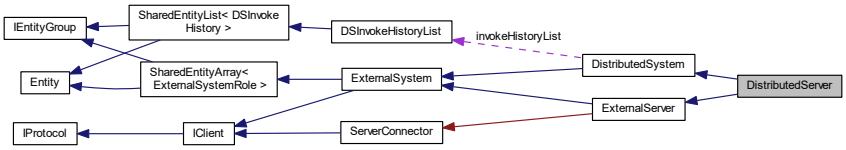
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSlaveClientMediator.hpp

6.23 DistributedServer Class Reference

A network driver for a distributed server.

`#include <DistributedServer.hpp>`

Collaboration diagram for DistributedServer:



Public Member Functions

- DistributedServer ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Additional Inherited Members

6.23.1 Detailed Description

A network driver for a distributed server.

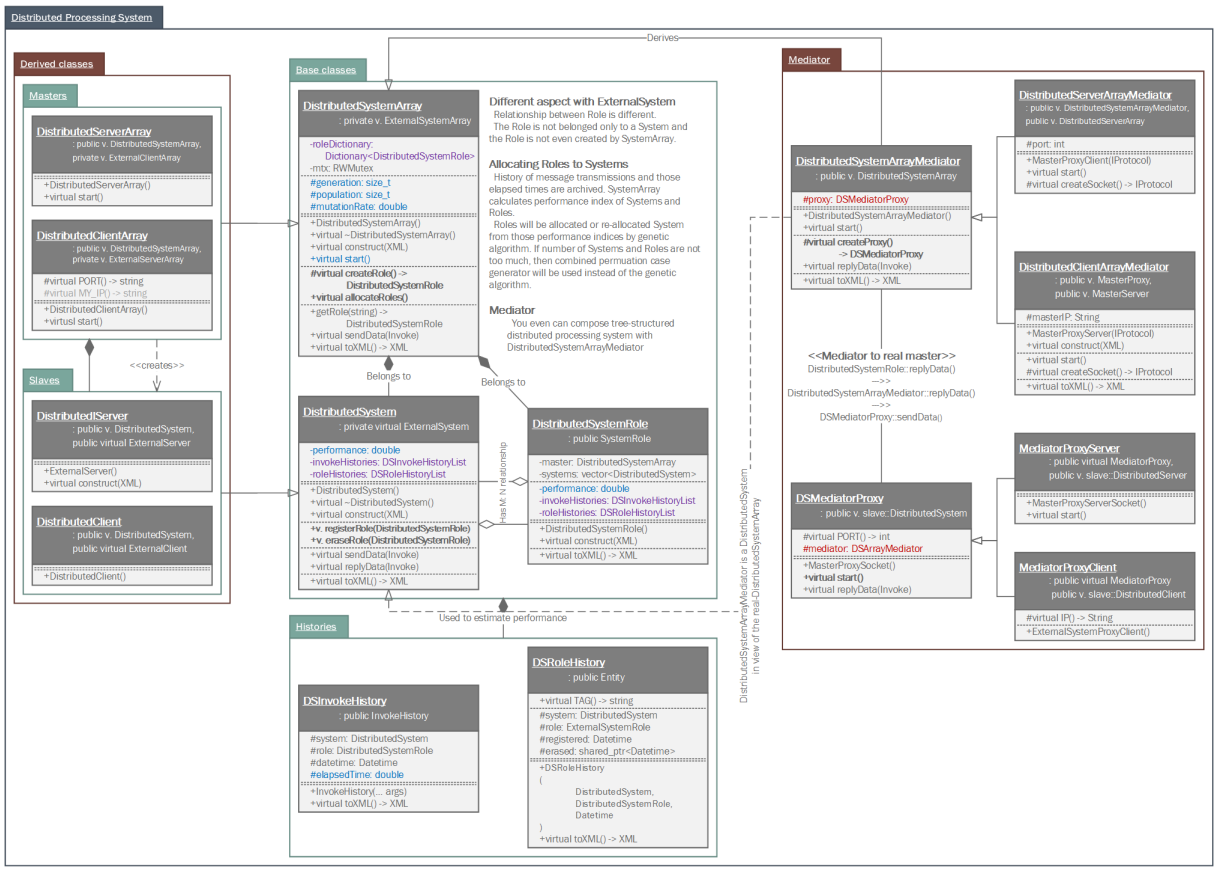
DistributedServer is an DistributedSystem specialized in server driver.

[Inherited]

DistributedSystem class is an ExternalSystem having role as DistributedSystemRole objects with performance index and history log allocation of DistributedSystemRole objects and have requsted Invoke message to related external system.

The greatest difference between DistributedSystem and ExternalSystem is, unlike the ExternalSystem class which having is ExternalSystemRole objects exclusively, DistributedSystem does not monopoly a DistributedSystemRole. The DistributedSystemRole can allocatedto multiple DistributedSystem.

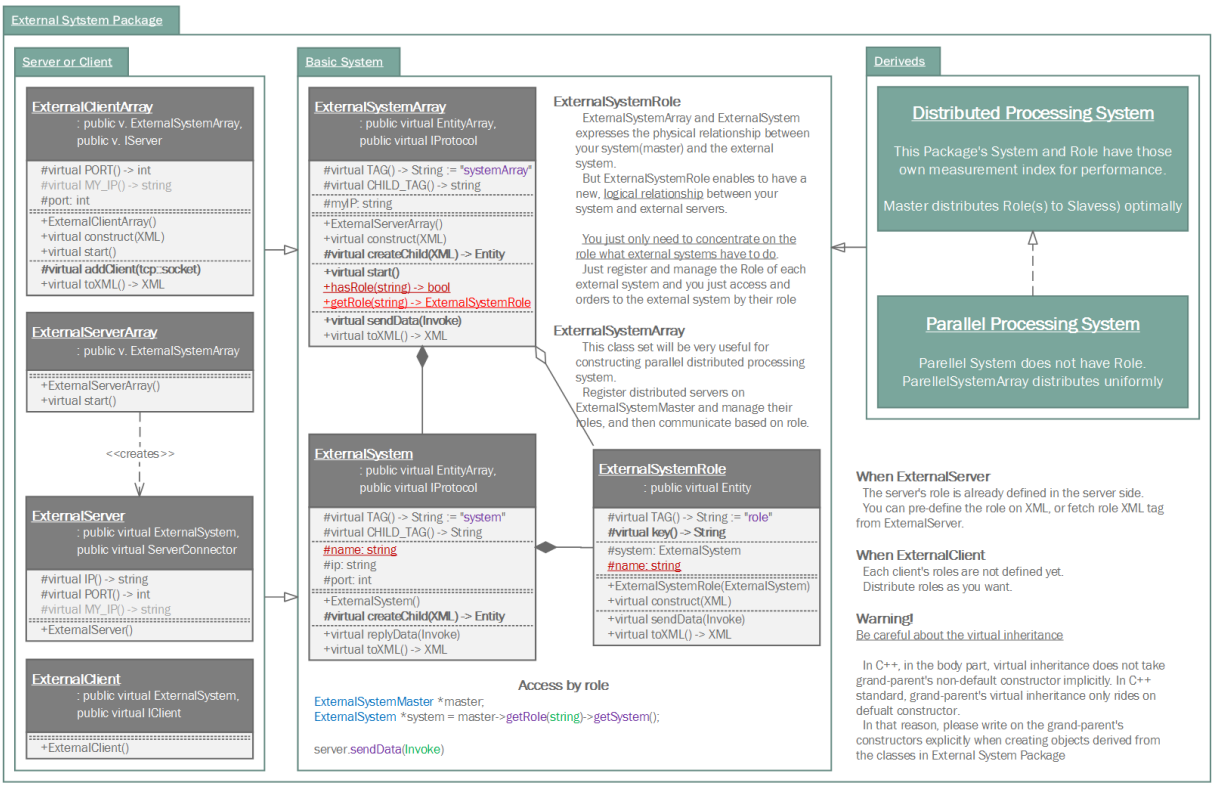
DistributedSystem and DistributedSystemRole classes have performance index and history log archiving what they've requested to the related distributed system. Allocating DistributedSystemRole objects to DistributedSyste will be determined those performance index and history log for optimize distribution of elasped time about each Invoke message processing.



[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, External↩System is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩ SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

6.23.2 Member Function Documentation

6.23.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from ExternalServer.

6.23.2.2 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ExternalServer.

The documentation for this class was generated from the following file:

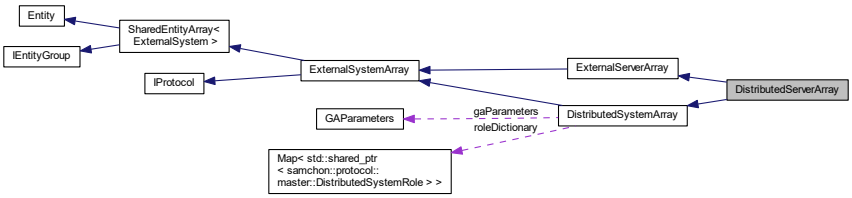
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedServer.hpp

6.24 DistributedServerArray Class Reference

An array of DistributedServer(s).

```
#include <DistributedServerArray.hpp>
```

Collaboration diagram for DistributedServerArray:



Public Member Functions

- DistributedServerArray ()

  Default Constructor.

- virtual void start () override

  Start interaction.

Additional Inherited Members

6.24.1 Detailed Description

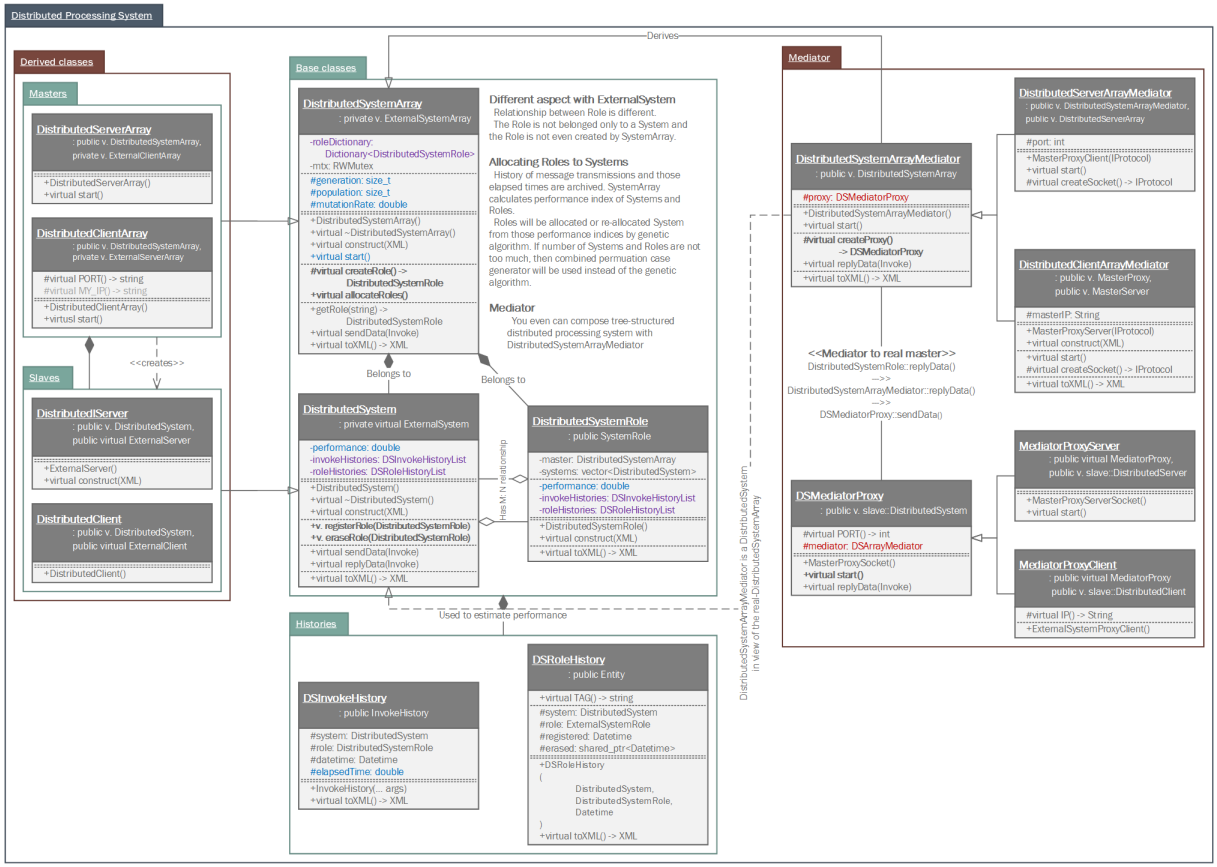
An array of DistributedServer(s).

DistributedServerArray is an DistributedSystemArray specialized in servers' driver.

[Inherited]

DistributedSystemArray class is an ExternalSystemArray and an abstract class containing and managing distributed system drivers within framework of master. The DistributedSystemArray class allocates DistributedSystemRole objects to DistributedSystem objects for optimization.

Unlike the ExternalSystemRole class which is dependent on an ExternalSystem, DistributedSystemRole is not dependent on a DistributedSystem and even can be allocated to multiple DistributedSystem objects. Allocation of DistributedSystemRole(s) to DistributedSystem(s) is determined by estimation of performance index pursuing best distribution. If number of DistributedSystem and DistributedSystemRole objects is not a lot, optimize by combined permuation algorithm. Else the objects are a lot, the allocation will be optimized by genetic algorithm.



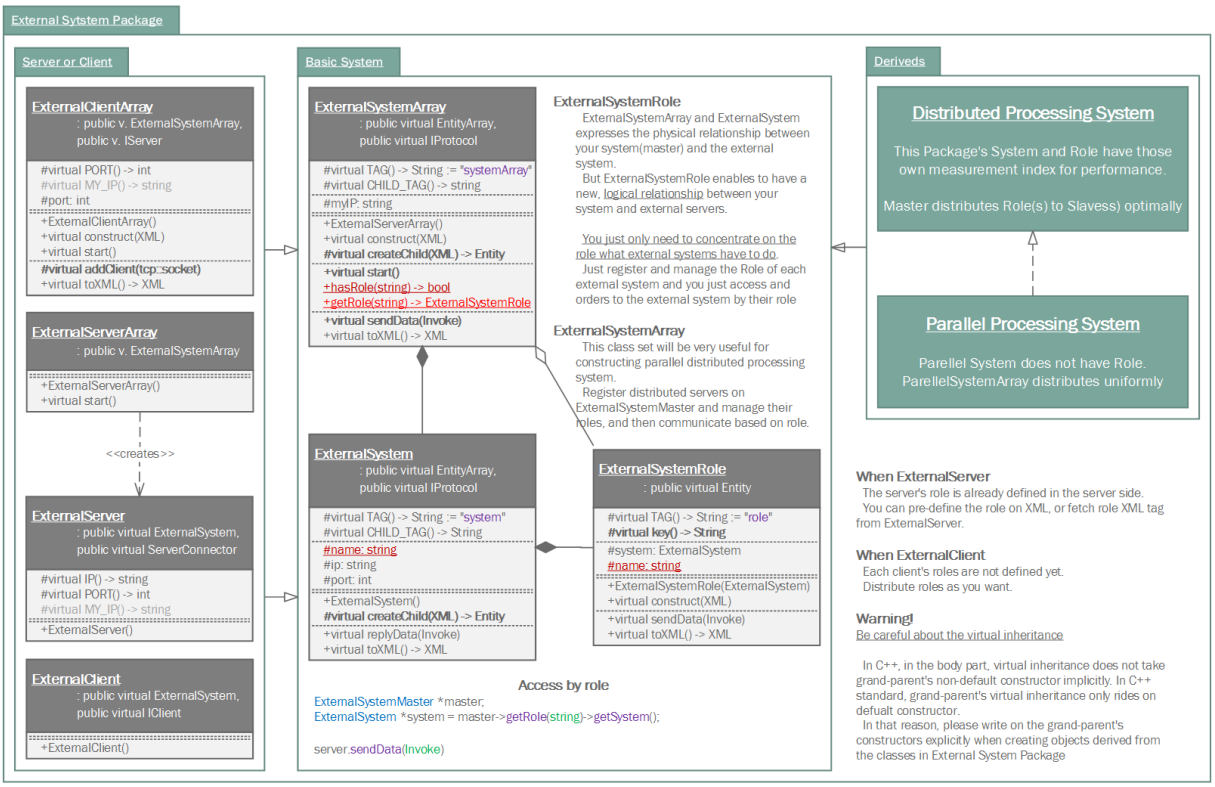
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.24.2 Member Function Documentation

6.24.2.1 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Reimplemented from ExternalServerArray.

Reimplemented in DistributedServerArrayMediator.

The documentation for this class was generated from the following file:

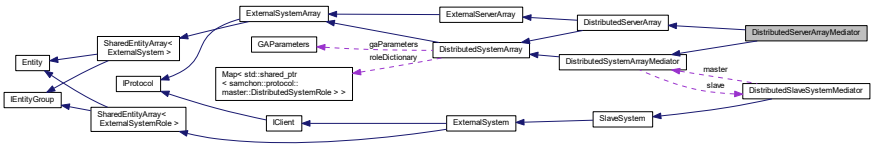
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedServerArray.hpp

6.25 DistributedServerArrayMediator Class Reference

A mediator of distributed system between master and slaves.

```
#include <DistributedServerArrayMediator.hpp>
```

Collaboration diagram for DistributedServerArrayMediator:



Public Member Functions

- DistributedServerArrayMediator ()

  Default Constructor.

- virtual void start () override

  Start interaction.

Additional Inherited Members

6.25.1 Detailed Description

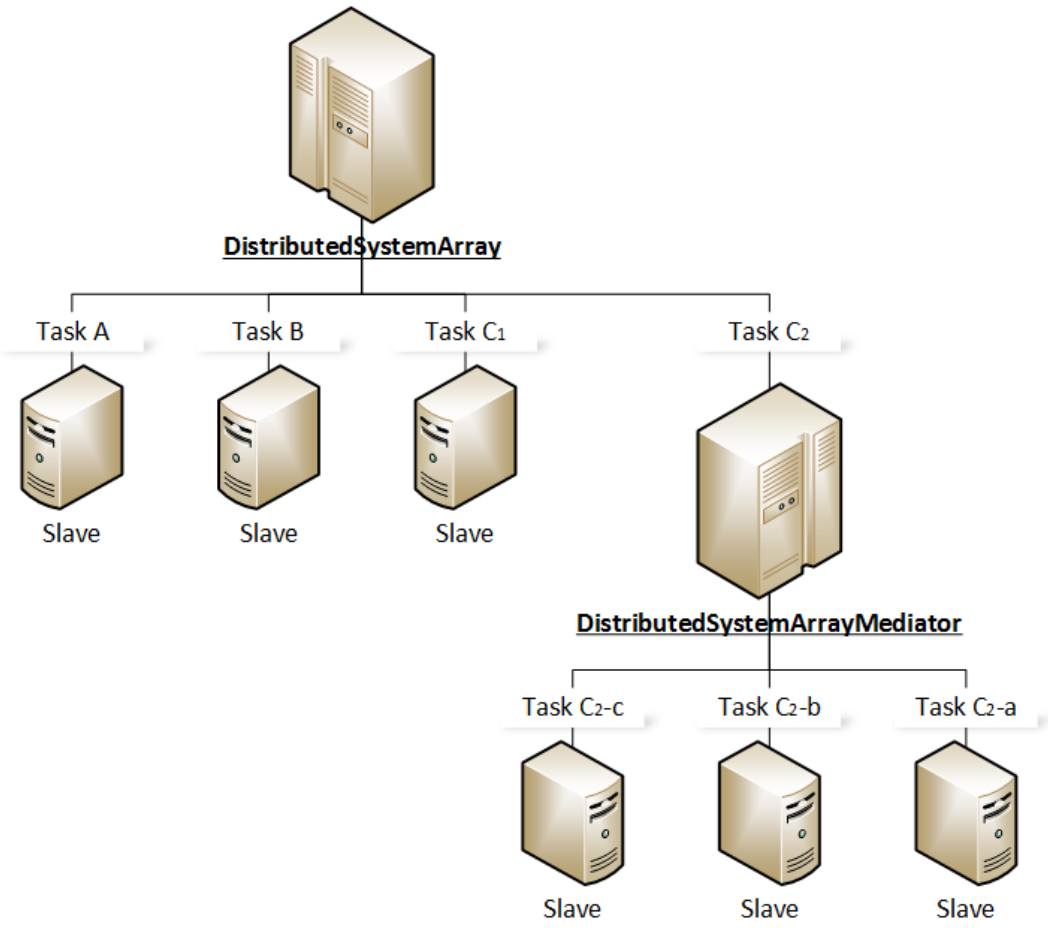
A mediator of distributed system between master and slaves.

DistributedServerArrayMediator is a DistributedSystemArrayMediator specialized in servers' driver. Distributed↩ ServerArrayMediator is a server for its master and also manages children servers' slaves.

[Inherited]

DistributedSystemArrayMediator class is used to realizing a mediator system of tree-structured distributed processing system. DistributedSystemArrayMediator intermediates between origin-master(DistributedSystemArray) and its slaves(slave::DistributedSystem).

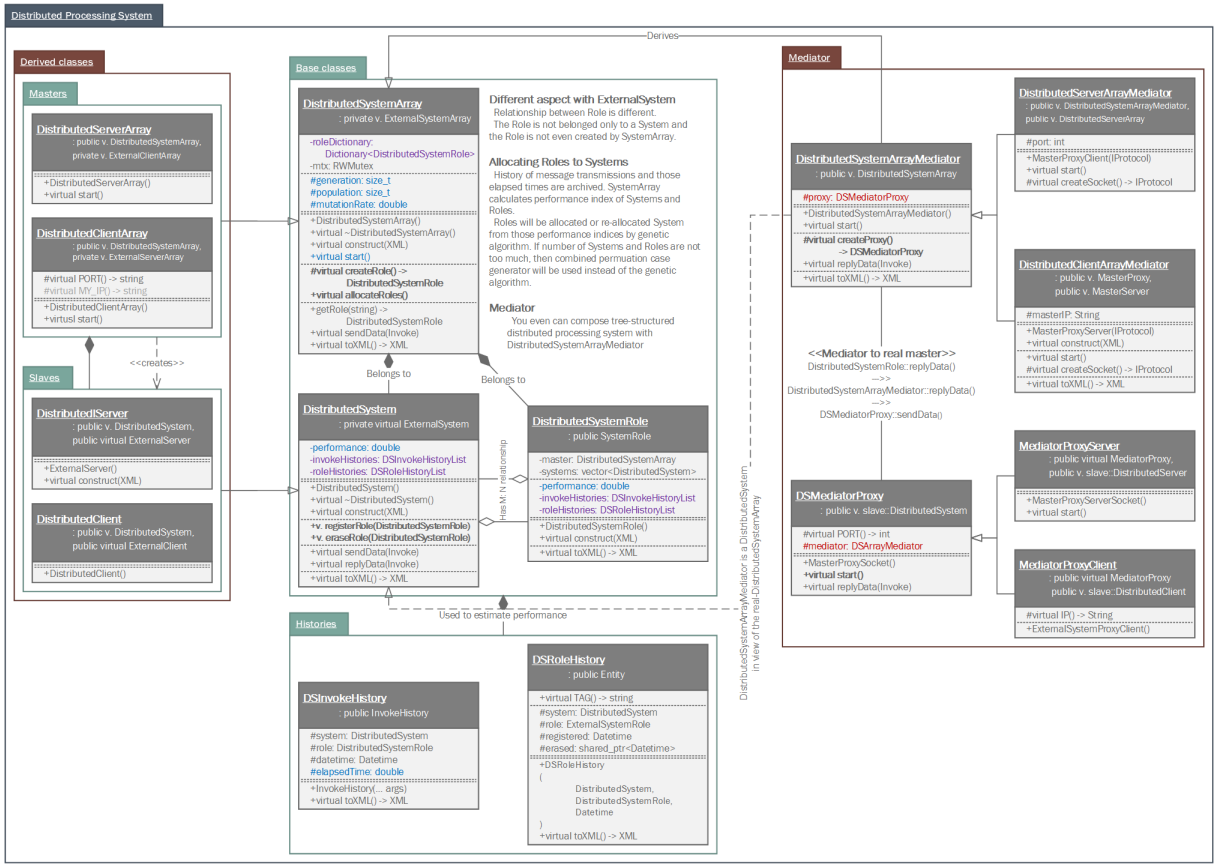
DistributedSystemArrayMediator is a master(DistributedSystemArray) within the framework of slaves systems (slave::DistributedSystem objects) and also can be a slave system (slave::DistributedSystem) within framework of its master(DistributedSystemArray) by having DistributedSlaveSystemMediator.



[Inherited]

DistributedSystemArray class is an ExternalSystemArray and an abstract class containing and managing distributed system drivers within framework of master. The DistributedSystemArray class allocates DistributedSystemRole objects to DistributedSystem objects for optimization.

Unlike the ExternalSystemRole class which is dependent on an ExternalSystem, DistributedSystemRole is not dependent on a DistributedSystem and even can be allocated to multiple DistributedSystem objects. Allocation of DistributedSystemRole(s) to DistributedSystem(s) is determined by estimation of performance index pursuing best distribution. If number of DistributedSystem and DistributedSystemRole objects is not a lot, optimize by combined permuation algorithm. Else the objects are a lot, the allocation will be optimized by genetic algorithm.



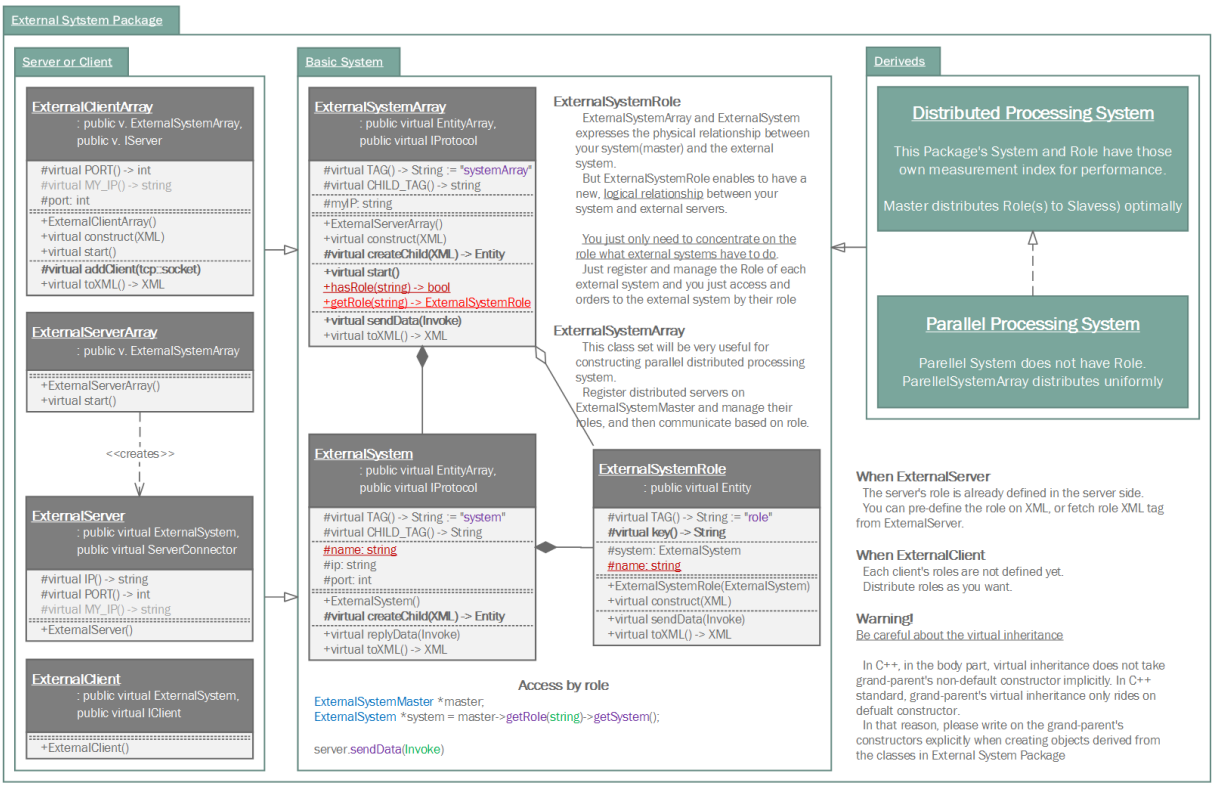
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.25.2 Member Function Documentation

6.25.2.1 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Reimplemented from DistributedServerArray.

The documentation for this class was generated from the following file:

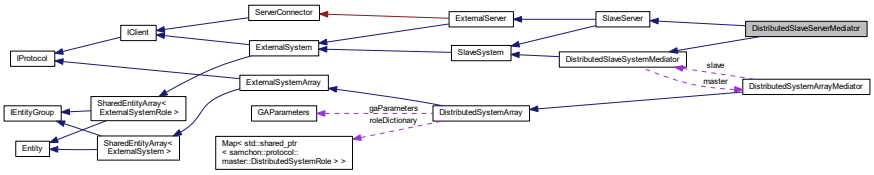
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedServerArrayMediator.↩ hpp

6.26 DistributedSlaveServerMediator Class Reference

A distributed slave server for mediation.

```
#include <DistributedSlaveServerMediator.hpp>
```

Collaboration diagram for DistributedSlaveServerMediator:



Public Member Functions

- DistributedSlaveServerMediator ()

  Default Constructor.

Additional Inherited Members

6.26.1 Detailed Description

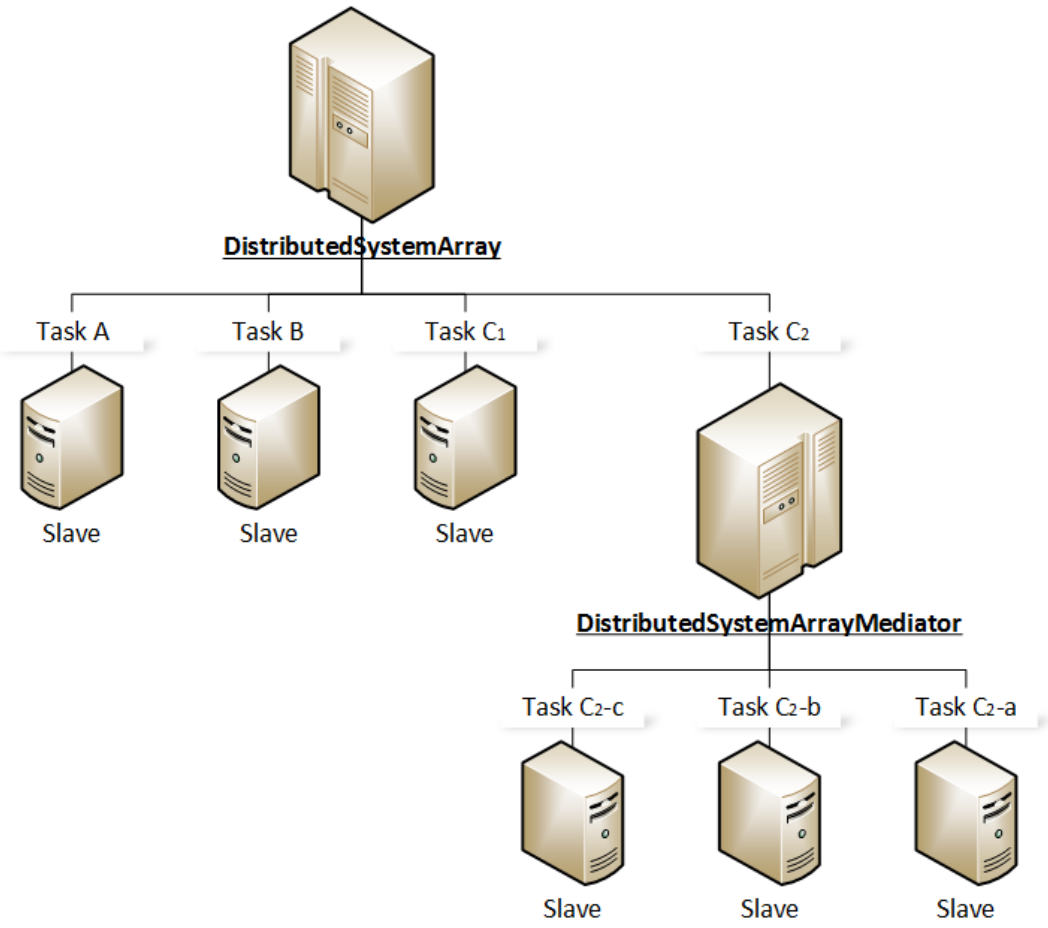
A distributed slave server for mediation.

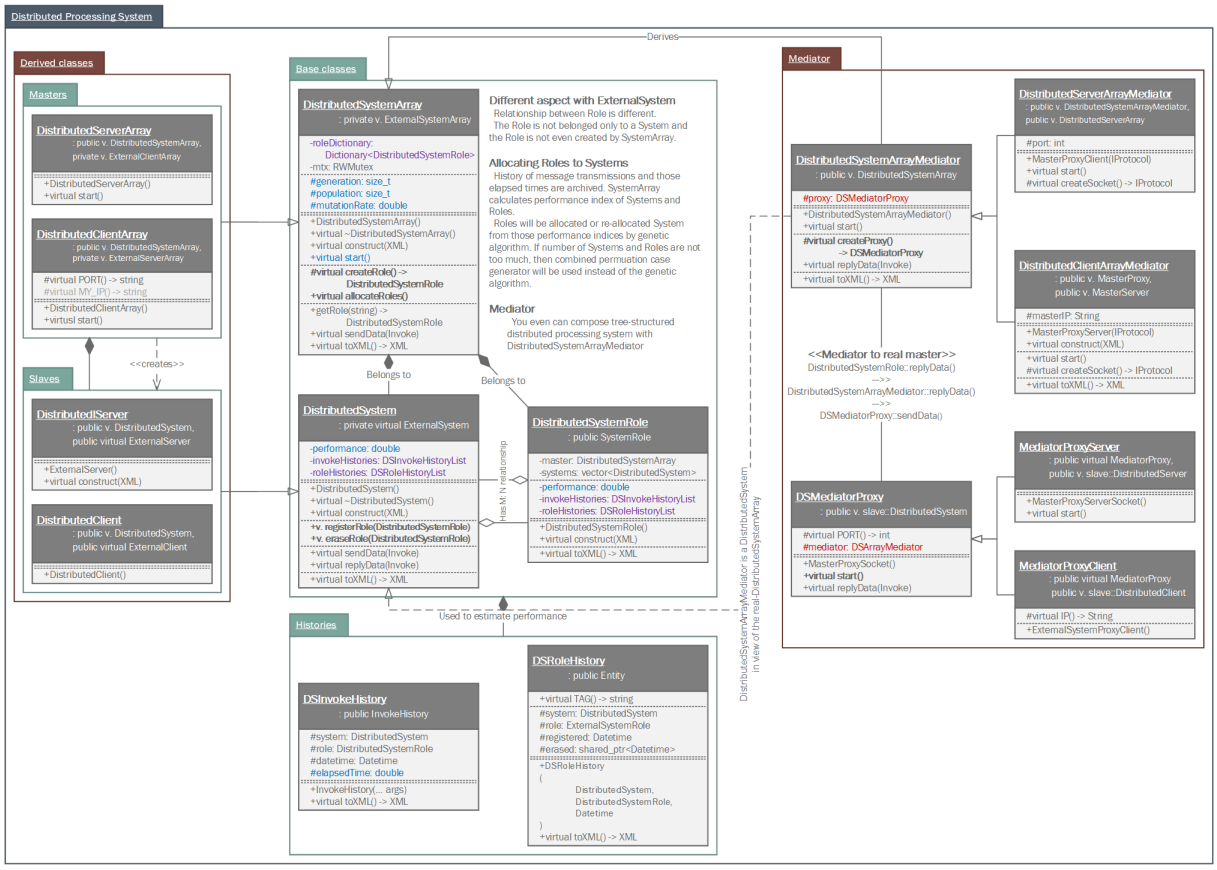
DistributedSlaveServerMediator is a DistributedSlaveSystemMediator class specialized in server driver.

[Inherited]

DistributedSlaveSystemMediator is an abstract class derived from slave::SlaveSystem for realizing tree-structured distriuted processins system. DistributedSlaveSystemMediator acts a slave system's role for its master, on the above level.

DistributedSystemArrayMediator and DistributedSlaveSystemMediator classes are paired to act a mediator system role by acting their own role, master system role for below slaves and slave system role for above master system.


DistributedSystemArray
Task A
Task B
Task C1
Task C2
Slave
Slave
Slave
DistributedSystemArrayMediator
Task C2-c
Task C2-b
Task C2-a
Slave
Slave
Slave


Distributed Processing System
Derived classes
Masters
DistributedServerArray
: public v. DistributedSystemArray, private v. ExternalClientArray
+DistributedServerArray()
+virtual start()
DistributedClientArray
: public v. DistributedSystemArray, private v. ExternalServerArray
#virtual PORT() -> string
#virtual MY_IP() -> string
+DistributedClientArray()
+virtual start()
<<creates>>
Slaves
DistributedIServer
: public v. DistributedSystem, public virtual ExternalServer
+ExternalServer()
+virtual construct(XML)
DistributedClient
: public v. DistributedSystem, public virtual ExternalClient
+DistributedClient()
Derives
Base classes
DistributedSystemArray
: private v. ExternalSystemArray
-roleDictionary: Dictionary<DistributedSystemRole>
-mtx: RWMutex
#generation: size_t
#population: size_t
#mutationRate: double
+DistributedSystemArray()
+virtual ~DistributedSystemArray()
+virtual construct(XML)
+virtual start()
#virtual createRole() -> DistributedSystemRole
+virtual allocateRoles()
+getRole(string) -> DistributedSystemRole
+virtual sendData(Invoke)
+virtual toXML() -> XML
Different aspect with ExternalSystem
Relationship between Role is different.
The Role is not belonged only to a System and the Role is not even created by SystemArray.
Allocating Roles to Systems
History of message transmissions and those elapsed times are archived. SystemArray calculates performance index of Systems and Roles.
Roles will be allocated or re-allocated System from those performance indices by genetic algorithm. If number of Systems and Roles are not too much, then combined permuation case generator will be used instead of the genetic algorithm.
Mediator
You even can compose tree-structured distributed processing system with DistributedSystemArrayMediator
Belongs to
Belongs to
DistributedSystem
: private virtual ExternalSystem
-performance: double
-invokeHistories: DSInvokeHistoryList
-roleHistories: DSRoleHistoryList
+DistributedSystem()
+virtual ~DistributedSystem()
+virtual construct(XML)
+v. registerRole(DistributedSystemRole)
+v. eraseRole(DistributedSystemRole)
+virtual sendData(Invoke)
+virtual replyData(Invoke)
+virtual toXML() -> XML
Has N, N relationship
DistributedSystemRole
: public SystemRole
-master: DistributedSystemArray
-systems: vector<DistributedSystem>
-performance: double
-invokeHistories: DSInvokeHistoryList
-roleHistories: DSRoleHistoryList
+DistributedSystemRole()
+virtual construct(XML)
+virtual toXML() -> XML
Used to estimate performance
Histories
DSInvokeHistory
: public InvokeHistory
#system: DistributedSystem
#role: DistributedSystemRole
#datetime: Datetime
#elapsedTime: double
+InvokeHistory(... args)
+virtual toXML() -> XML
DSRoleHistory
: public Entity
+virtual TAG() -> string
#system: DistributedSystem
#role: ExternalSystemRole
#registered: Datetime
#erased: shared_ptr<Datetime>
+DSRoleHistory (DistributedSystem, DistributedSystemRole, Datetime)
+virtual toXML() -> XML
DistributedSystemArrayMediator is a DistributedSystem in view of the real-DistributedSystemArray
Mediator
DistributedSystemArrayMediator
: public v. DistributedSystemArray
#proxy: DSMediatorProxy
+DistributedSystemArrayMediator()
+virtual start()
#virtual createProxy() -> DSMediatorProxy
+virtual replyData(Invoke)
+virtual toXML() -> XML
<<Mediator to real master>>
DistributedSystemRole::replyData()
-->>
DistributedSystemArrayMediator::replyData()
-->>
DSMediatorProxy::sendData()
DSMediatorProxy
: public v. slave::DistributedSystem
#virtual PORT() -> int
#mediator: DSArrayMediator
+MasterProxySocket()
+virtual start()
+virtual replyData(Invoke)
DistributedServerArrayMediator
: public v. DistributedSystemArrayMediator, public v. DistributedServerArray
#port: int
+MasterProxyClient(IProtocol)
+virtual start()
#virtual createSocket() -> IProtocol
DistributedClientArrayMediator
: public v. MasterProxy, public v. MasterServer
#masterIP: String
+MasterProxyServer(IProtocol)
+virtual construct(XML)
+virtual start()
#virtual createSocket() -> IProtocol
+virtual toXML() -> XML
MediatorProxyServer
: public virtual MediatorProxy, public v. slave::DistributedServer
+MasterProxyServerSocket()
+virtual start()
MediatorProxyClient
: public virtual MediatorProxy public v. slave::DistributedClient
#virtual IP() -> String
+ExternalSystemProxyClient()

[Inherited]

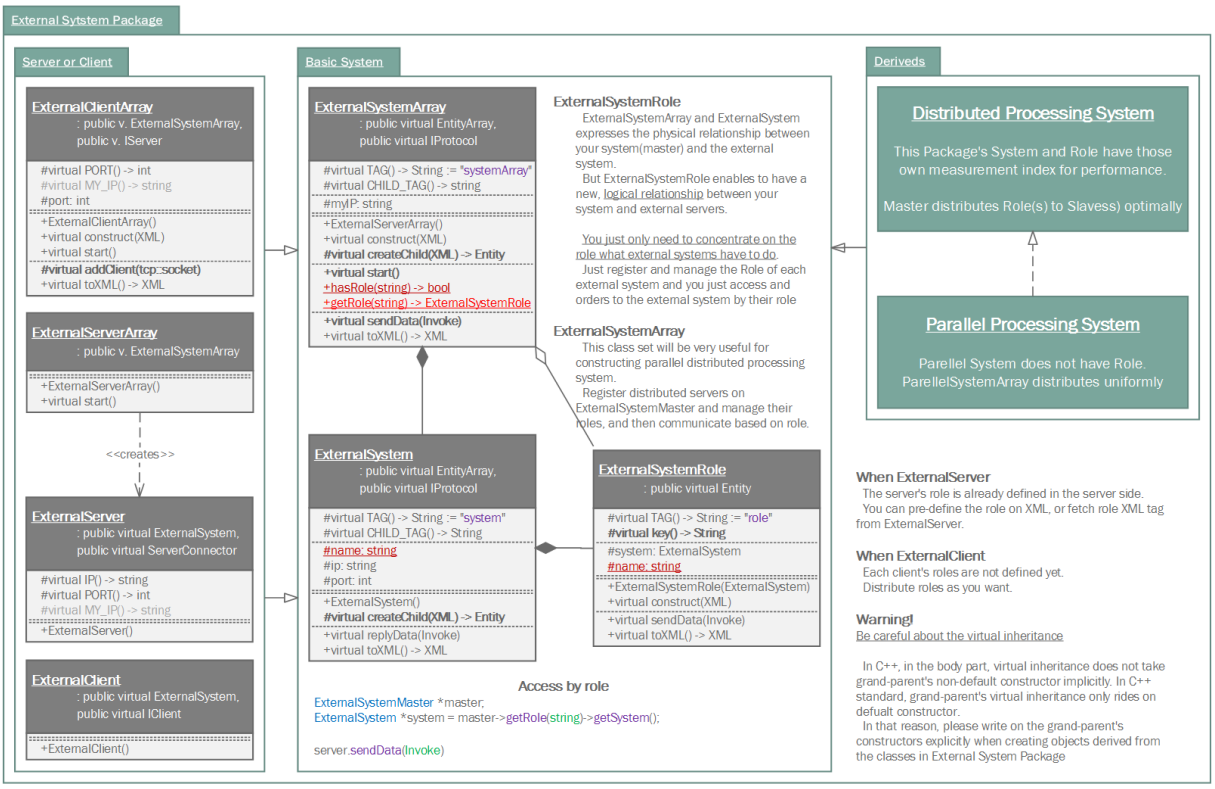
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::ParallelSystem and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, External↩System is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

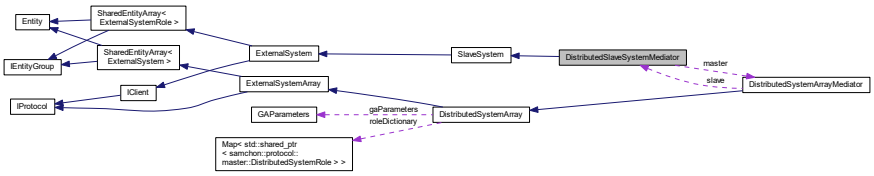
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSlaveServerMediator.↩ hpp

6.27 DistributedSlaveSystemMediator Class Reference

A distributed slave system for mediation.

```
#include <DistributedSlaveSystemMediator.hpp>
```

Collaboration diagram for DistributedSlaveSystemMediator:



Public Member Functions

- DistributedSlaveSystemMediator ()

  Default Constructor.
- virtual void replyData (std::shared_ptr< Invoke >) override

  Handling replied message from an external system.
- virtual auto TAG () const -> std::string override

  A tag name when represented by XML.

Protected Attributes

- DistributedSystemArrayMediator * master

  A master system for mediation.

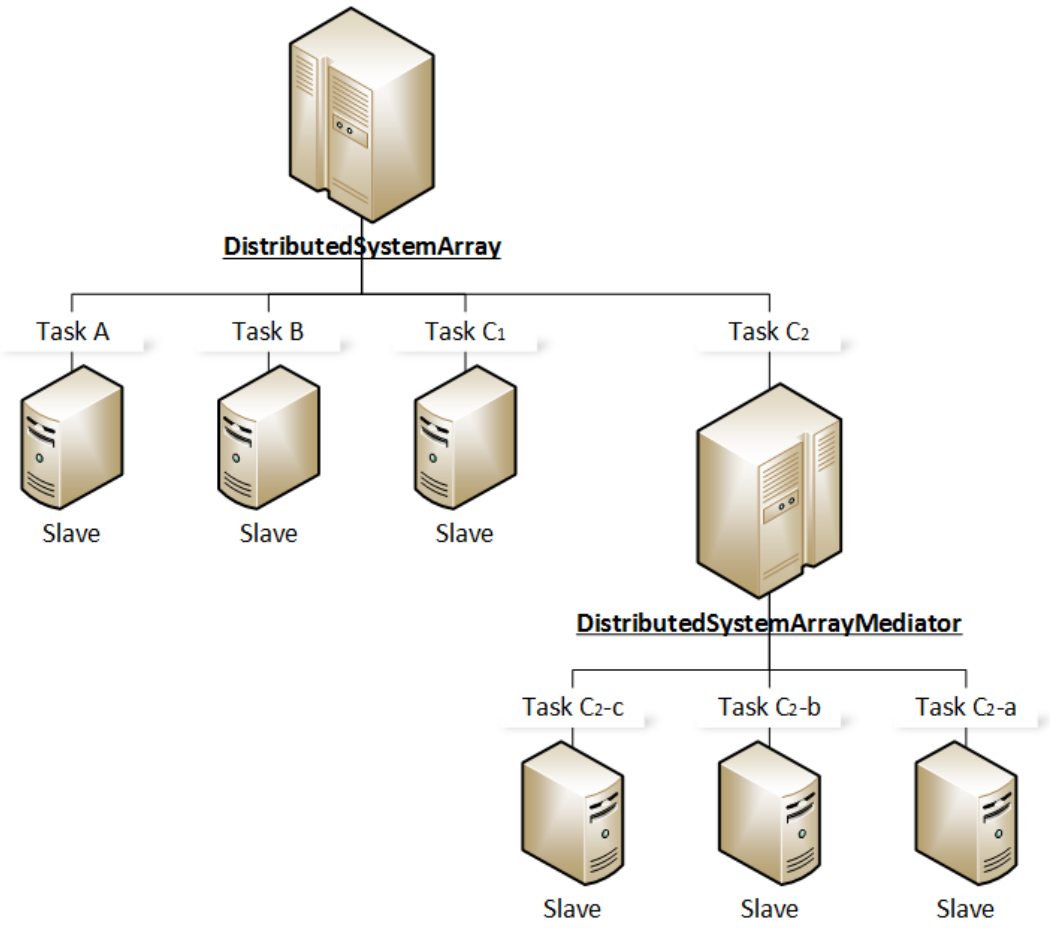
Additional Inherited Members

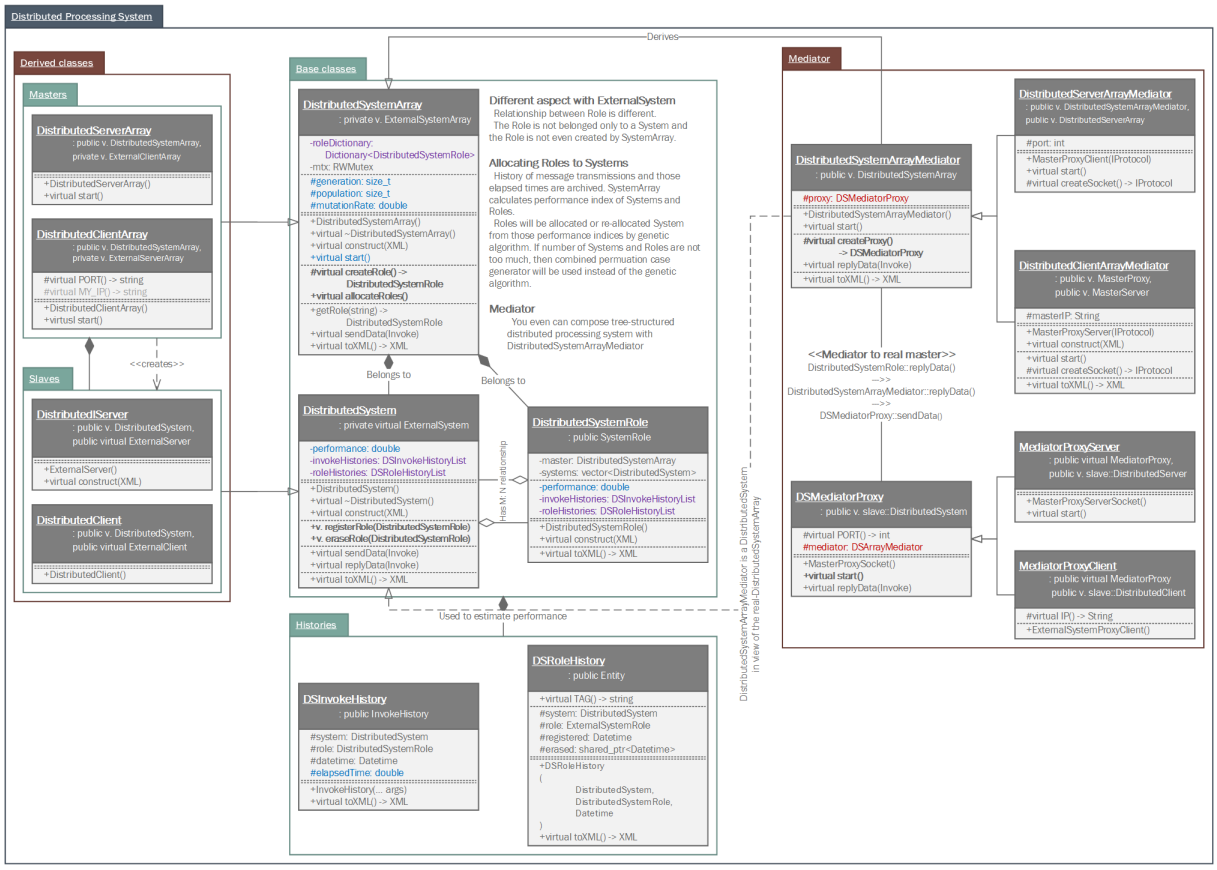
6.27.1 Detailed Description

A distributed slave system for mediation.

DistributedSlaveSystemMediator is an abstract class derived from slave::SlaveSystem for realizing tree-structured distriuted processins system. DistributedSlaveSystemMediator acts a slave system's role for its master, on the above level.

DistributedSystemArrayMediator and DistributedSlaveSystemMediator classes are paired to act a mediator system role by acting their own role, master system role for below slaves and slave system role for above master system.


DistributedSystemArray
Task A
Task B
Task C1
Task C2
Slave
Slave
Slave
DistributedSystemArrayMediator
Task C2-c
Task C2-b
Task C2-a
Slave
Slave
Slave


Distributed Processing System
Derived classes
Masters
DistributedServerArray
: public v. DistributedSystemArray,
private v. ExternalClientArray
+DistributedServerArray()
+virtual start()
DistributedClientArray
: public v. DistributedSystemArray,
private v. ExternalServerArray
#virtual PORT() -> string
#virtual MY_IP() -> string
+DistributedClientArray()
+virtual start()
<<creates>>
Slaves
DistributedIServer
: public v. DistributedSystem,
public virtual ExternalServer
+ExternalServer()
+virtual construct(XML)
DistributedClient
: public v. DistributedSystem,
public virtual ExternalClient
+DistributedClient()
Derives
Base classes
DistributedSystemArray
: private v. ExternalSystemArray
-roleDictionary:
Dictionary<DistributedSystemRole>
-mtx: RWMutex
#generation: size_t
#population: size_t
#mutationRate: double
+DistributedSystemArray()
+virtual ~DistributedSystemArray()
+virtual construct(XML)
+virtual start()
#virtual createRole() ->
DistributedSystemRole
+virtual allocateRoles()
+getRole(string) ->
DistributedSystemRole
+virtual sendData(Invoke)
+virtual toXML() -> XML
Different aspect with ExternalSystem
Relationship between Role is different.
The Role is not belonged only to a System and the Role is not even created by SystemArray.
Allocating Roles to Systems
History of message transmissions and those elapsed times are archived. SystemArray calculates performance index of Systems and Roles.
Roles will be allocated or re-allocated System from those performance indices by genetic algorithm. If number of Systems and Roles are not too much, then combined permuation case generator will be used instead of the genetic algorithm.
Mediator
You even can compose tree-structured distributed processing system with DistributedSystemArrayMediator
Belongs to
Belongs to
DistributedSystem
: private virtual ExternalSystem
-performance: double
-invokeHistories: DSInvokeHistoryList
-roleHistories: DSRoleHistoryList
+DistributedSystem()
+virtual ~DistributedSystem()
+virtual construct(XML)
+v. registerRole(DistributedSystemRole)
+v. eraseRole(DistributedSystemRole)
+virtual sendData(Invoke)
+virtual replyData(Invoke)
+virtual toXML() -> XML
Has N:N relationship
DistributedSystemRole
: public SystemRole
-master: DistributedSystemArray
-systems: vector<DistributedSystem>
-performance: double
-invokeHistories: DSInvokeHistoryList
-roleHistories: DSRoleHistoryList
+DistributedSystemRole()
+virtual construct(XML)
+virtual toXML() -> XML
Used to estimate performance
Histories
DSInvokeHistory
: public InvokeHistory
#system: DistributedSystem
#role: DistributedSystemRole
#datetime: Datetime
#elapsedTime: double
+InvokeHistory(... args)
+virtual toXML() -> XML
DSRoleHistory
: public Entity
+virtual TAG() -> string
#system: DistributedSystem
#role: ExternalSystemRole
#registered: Datetime
#erased: shared_ptr<Datetime>
+DSRoleHistory
(
DistributedSystem,
DistributedSystemRole,
Datetime
)
+virtual toXML() -> XML
DistributedSystemArrayMediator is a DistributedSystem in view of the real-DistributedSystemArray
Mediator
DistributedSystemArrayMediator
: public v. DistributedSystemArray
#proxy: DSMediatorProxy
+DistributedSystemArrayMediator()
+virtual start()
#virtual createProxy()
-> DSMediatorProxy
+virtual replyData(Invoke)
+virtual toXML() -> XML
<<Mediator to real master>>
DistributedSystemRole::replyData()
-->>
DistributedSystemArrayMediator::replyData()
-->>
DSMediatorProxy::sendData()
DSMediatorProxy
: public v. slave::DistributedSystem
#virtual PORT() -> int
#mediator: DSArrayMediator
+MasterProxySocket()
+virtual start()
+virtual replyData(Invoke)
DistributedServerArrayMediator
: public v. DistributedSystemArrayMediator,
public v. DistributedServerArray
#port: int
+MasterProxyClient(IProtocol)
+virtual start()
#virtual createSocket() -> IProtocol
DistributedClientArrayMediator
: public v. MasterProxy,
public v. MasterServer
#masterIP: String
+MasterProxyServer(IProtocol)
+virtual construct(XML)
+virtual start()
#virtual createSocket() -> IProtocol
+virtual toXML() -> XML
MediatorProxyServer
: public virtual MediatorProxy,
public v. slave::DistributedServer
+MasterProxyServerSocket()
+virtual start()
MediatorProxyClient
: public virtual MediatorProxy
public v. slave::DistributedClient
#virtual IP() -> String
+ExternalSystemProxyClient()

[Inherited]

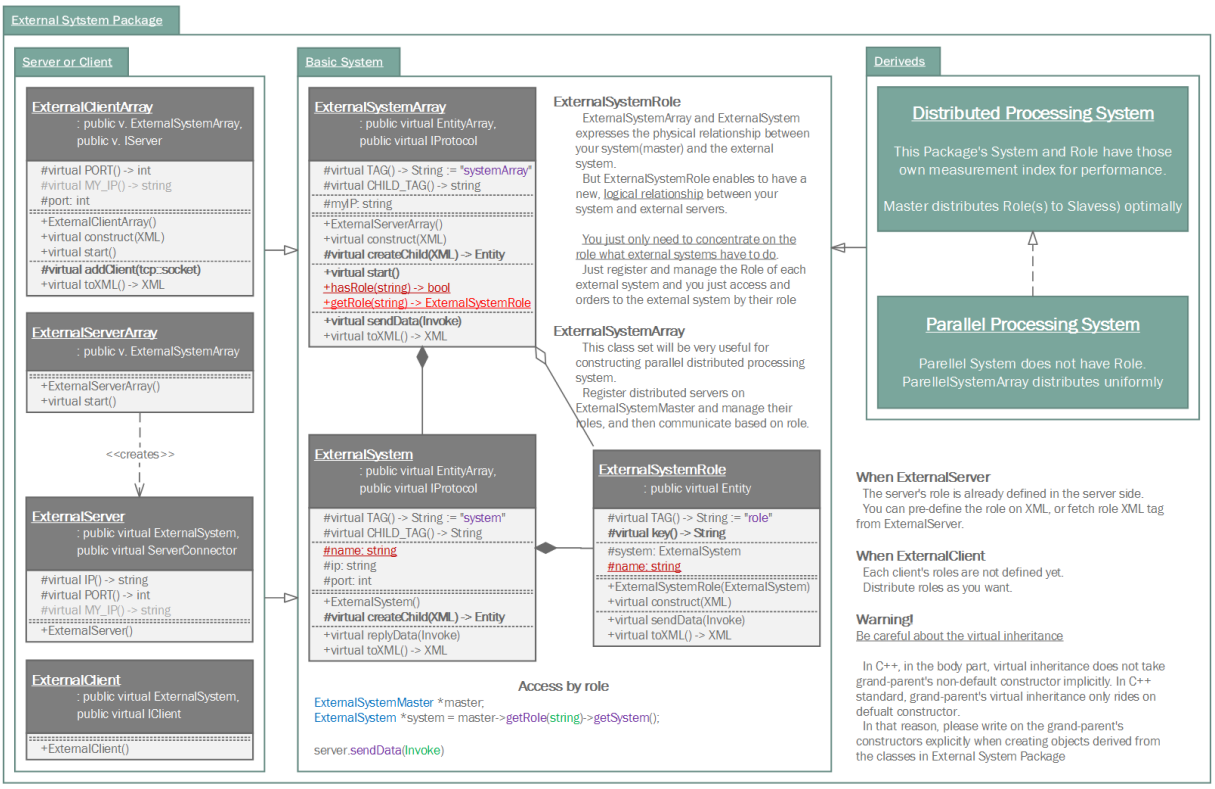
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::ParallelSystem and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, External↩System is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

6.27.2 Member Function Documentation

6.27.2.1 virtual void replyData (std::shared_ptr< Invoke >) [override], [virtual]

Handling replied message from an external system.

If there's a related ExternalSystemRole, shifts responsibility to the ExternalSystemRole.

Parameters

invoke	Replied Invoke message

Reimplemented from ExternalSystem.

6.27.2.2 virtual auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Reimplemented from ExternalSystem.

The documentation for this class was generated from the following file:

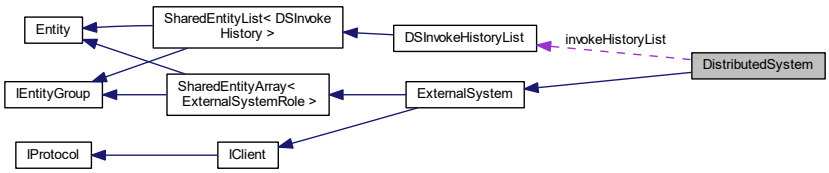
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSlaveSystemMediator.hpp

6.28 DistributedSystem Class Reference

A network driver for a distributed system.

```
#include <DistributedSystem.hpp>
```

Collaboration diagram for DistributedSystem:



Public Member Functions

- DistributedSystem ()

Default Constructor.

- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.

- virtual void sendData (std::shared_ptr< Invoke >) override

  Sends message to a related system.

- virtual void replyData (std::shared_ptr< Invoke >) override

  Handling replied message from an external system.

- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Protected Attributes

- size_t inProgress

  A number of processes running on the distributed system.

- size_t processed

  A number of processed have runned.

- double avgElapsedTime

  Average of elapsed time.

- double performance

  A performance index.

- DSRoleHistoryList * roleHistoryList

  A list of history log for role allocations.

- DSInvokeHistoryList * invokeHistoryList

  A list of history log for reported Invoke messages.

Additional Inherited Members

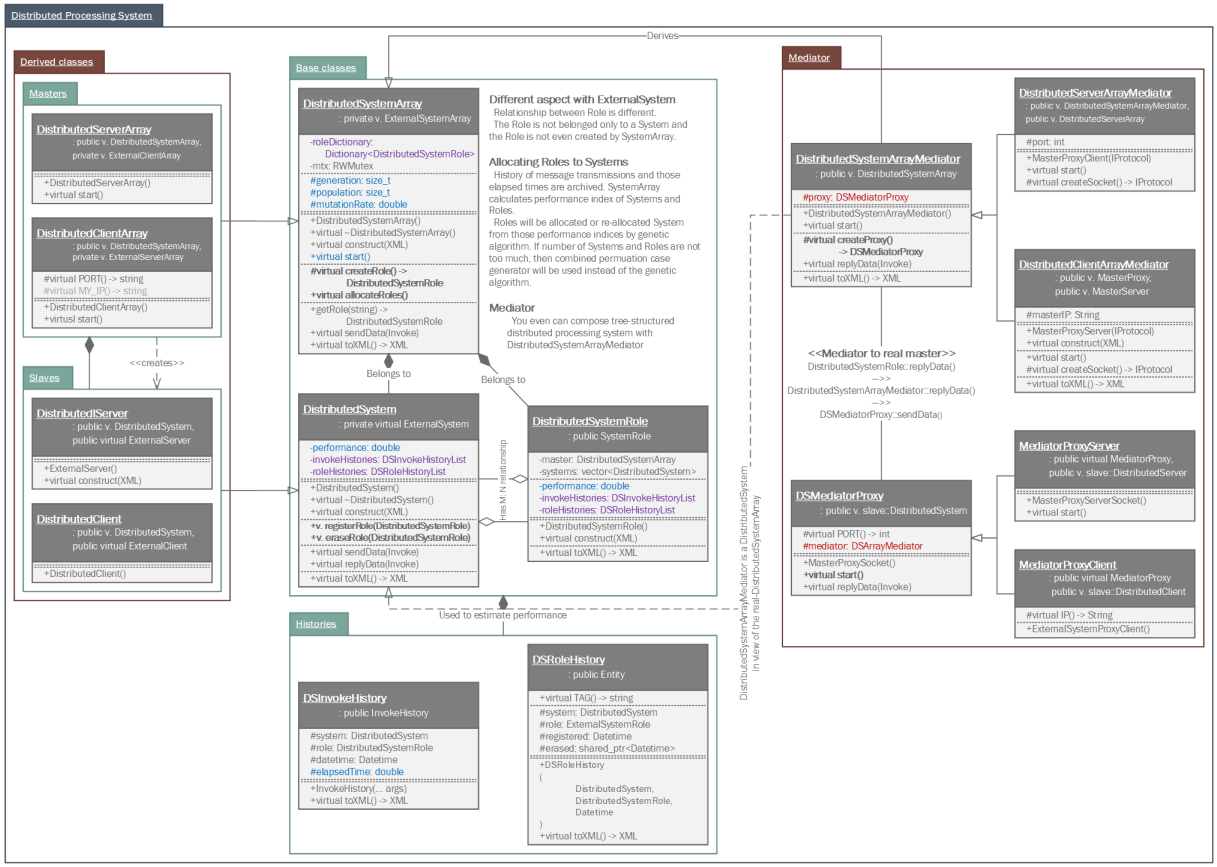
6.28.1 Detailed Description

A network driver for a distributed system.

DistributedSystem class is an ExternalSystem having role as DistributedSystemRole objects with performance index and history log allocation of DistributedSystemRole objects and have requsted Invoke message to related external system.

The greatest difference between DistributedSystem and ExternalSystem is, unlike the ExternalSystem class which having is ExternalSystemRole objects exclusively, DistributedSystem does not monopoly a DistributedSystemRole. The DistributedSystemRole can allocatedto multiple DistributedSystem.

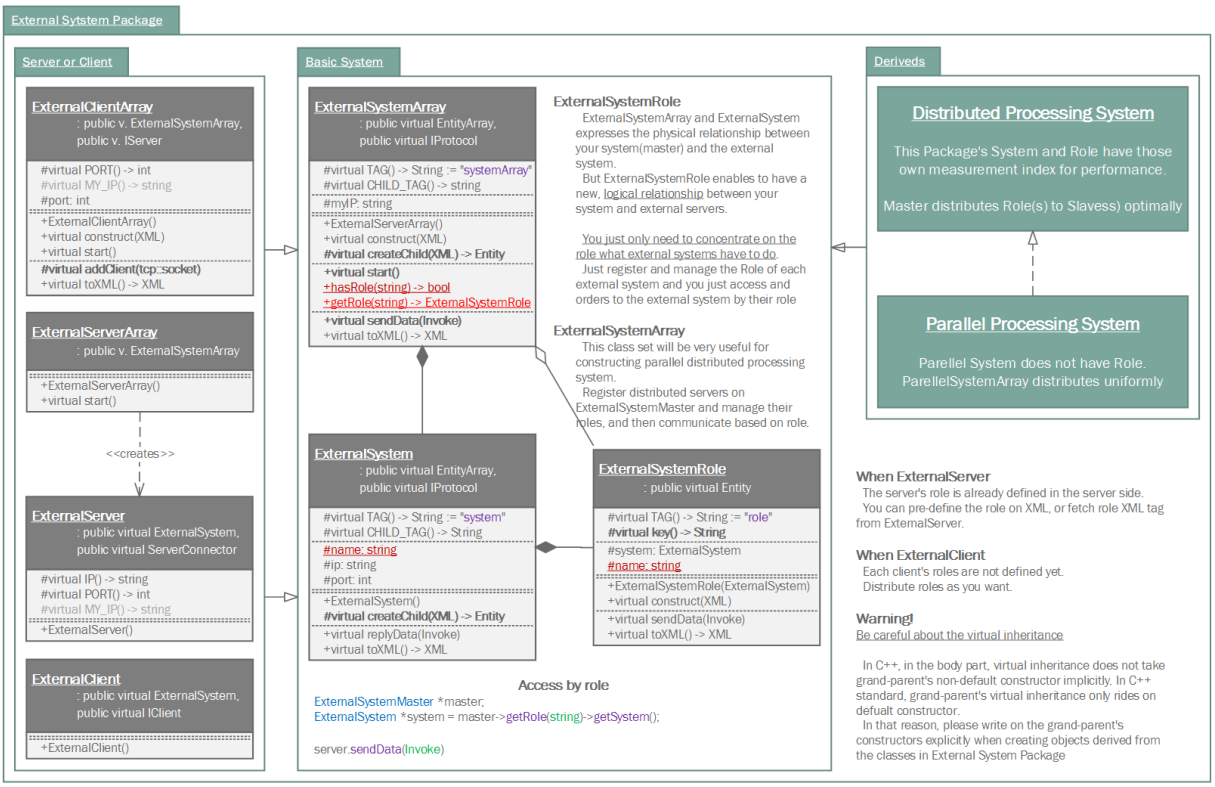
DistributedSystem and DistributedSystemRole classes have performance index and history log archiving what they've requested to the related distributed system. Allocating DistributedSystemRole objects to DistributedSyste will be determined those performance index and history log for optimize distribution of elasped time about each Invoke message processing.



[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, External↩System is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩ SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

6.28.2 Member Function Documentation

6.28.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from ExternalSystem.

Reimplemented in DistributedServer.

6.28.2.2 virtual void sendData (std::shared_ptr< Invoke >) [override], [virtual]

Sends message to a related system.

Parameters

invoke	Invoke message to send

Reimplemented from IClient.

6.28.2.3 virtual void replyData (std::shared_ptr< Invoke >) [override], [virtual]

Handling replied message from an external system.

If there's a related ExternalSystemRole, shifts responsibility to the ExternalSystemRole.

Parameters

invoke	Replied Invoke message

Reimplemented from ExternalSystem.

6.28.2.4 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ExternalSystem.

Reimplemented in DistributedServer.

6.28.3 Member Data Documentation

6.28.3.1 size_t inProgress [protected]

A number of processes running on the distributed system.

Adds one count when sendData() is called and minus one count if a DSInvokeHistory, which means reported Invoke history, was reported by replyData().

If an DistributedSystemRole is allocated to multiple DistributedSystem objects, which DistributedSystem will be used is determined by the *process size* and performance index of the ExternalSystemRole.

6.28.3.2 double avgElapsedTime [protected]

Average of elapsed time.

Average elapsed time of handling Invoke message (request) in frame work of tbe related distributed system. Not in frame work of distributed system's driver, DistributeDSystem.

The ultimate optimization target of distributed processing system is to minimizing standard deviation of the average elapsed time of the DistributedSystem.

6.28.3.3 double performance [protected]

A performance index.

A performance index z is calculated by normalization calculating reverse number of whole distributed system's average elapsed times and its z value between the normal distribution.

- X = (

  If a distributed system has no history of handling Invoke message, then set the performance index to 0 as default. The performance index will be re-calcuated whenever re-allocation of roles is done (DistributedSystem::allocateRoles()).

  The documentation for this class was generated from the following file:

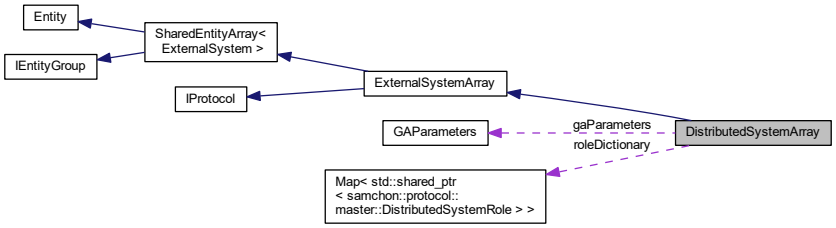
  - D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSystem.hpp

6.29 DistributedSystemArray Class Reference

An array of distributed system drivers.

```
#include <DistributedSystemArray.hpp>
```

Collaboration diagram for DistributedSystemArray:



Public Member Functions

- DistributedSystemArray ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual void start () override

  Start interaction.
- virtual void allocateRoles ()

  Allocate roles to each distributed processing system.
- auto getRole (const std::string &) const -> std::shared_ptr< DistributedSystemRole >

  Get a role.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Protected Member Functions

- virtual auto createRole (std::shared_ptr< library::XML >) -> DistributedSystemRole *=0

  Factory method of a DistributedSystemRole.

Protected Attributes

- Dictionary< std::shared_ptr< DistributedSystemRole > > roleDictionary

  A dictionary of role objects.
- struct library::GAParameters gaParameters

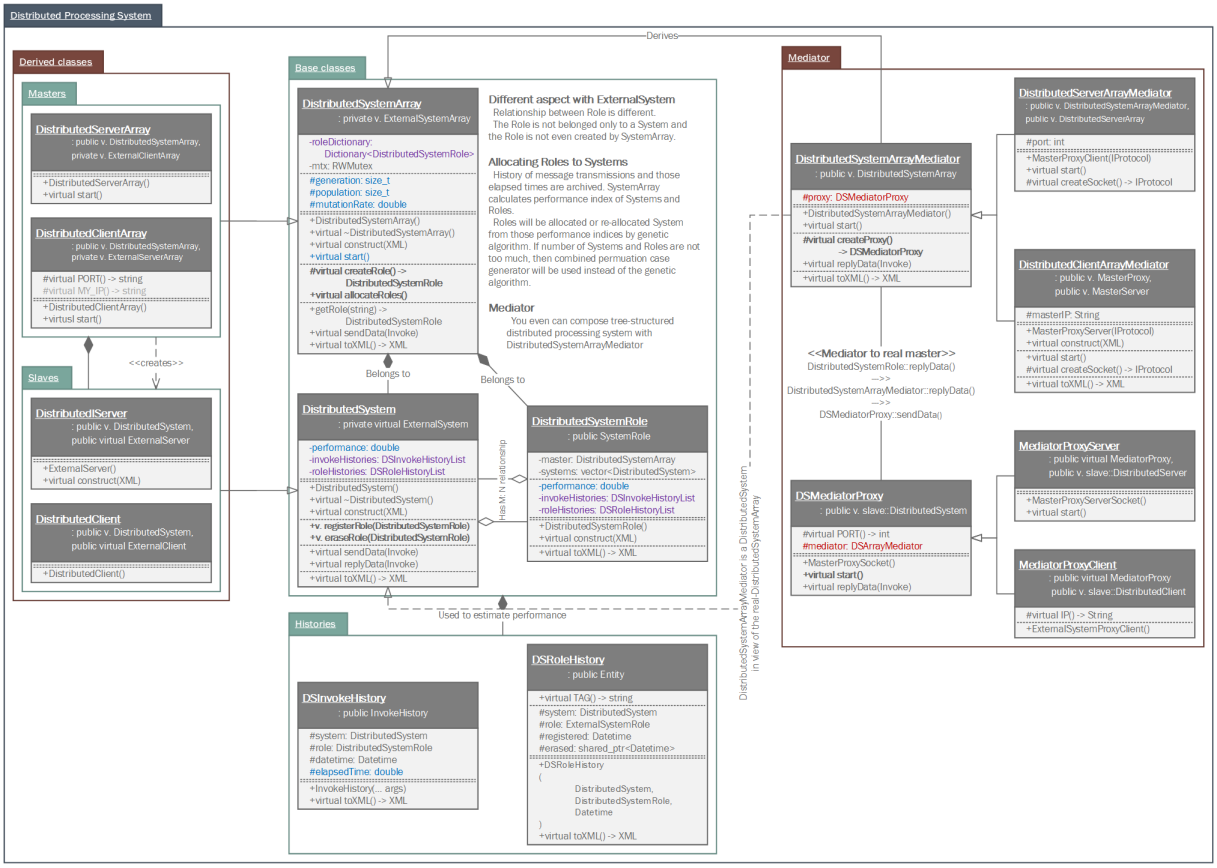
  A structure of parameters for genetic algorithm.

6.29.1 Detailed Description

An array of distributed system drivers.

DistributedSystemArray class is an ExternalSystemArray and an abstract class containing and managing distributed system drivers within framework of master. The DistributedSystemArray class allocates DistributedSystemRole objects to DistributedSystem objects for optimization.

Unlike the ExternalSystemRole class which is dependent on an ExternalSystem, DistributedSystemRole is not dependent on a DistributedSystem and even can be allocated to multiple DistributedSystem objects. Allocation of DistributedSystemRole(s) to DistributedSystem(s) is determined by estimation of performance index pursuing best distribution. If number of DistributedSystem and DistributedSystemRole objects is not a lot, optimize by combined permuation algorithm. Else the objects are a lot, the allocation will be optimized by genetic algorithm.



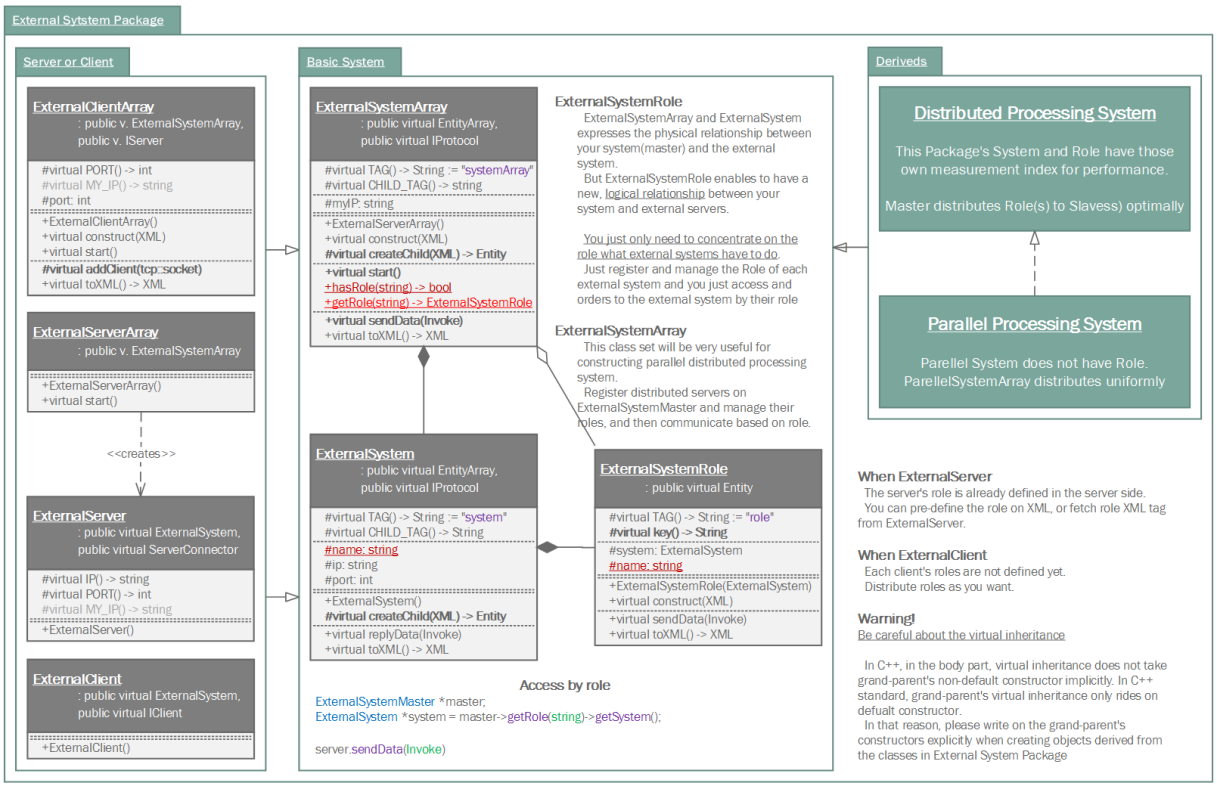
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.29.2 Member Function Documentation

6.29.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in DistributedSystemArrayMediator, DistributedClientArrayMediator, and DistributedClient↩Array.

6.29.2.2 virtual auto createRole (std::shared_ptr< library::XML >) -> DistributedSystemRole * [protected], [pure virtual]

Factory method of a DistributedSystemRole.

Parameters

An	XML object represents the role.

Returns

A new DistributedSystemRole.

6.29.2.3 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Implements ExternalSystemArray.

Reimplemented in DistributedSystemArrayMediator, DistributedClientArrayMediator, DistributedClientArray, DistributedServerArrayMediator, and DistributedServerArray.

6.29.2.4 virtual void allocateRoles ( ) [virtual]

Allocate roles to each distributed processing system.

Allocates DistributedSystemRole objects to Distributedsystem objects. The allocation will be done when:

- New distributed system (client) has connected.
- Ordinary distributed system is disconnected.
- Deviation of elapsed time of whole DistributedSystem overs limited value.

6.29.2.5 auto getRole (const std::string &) const -> std::shared_ptr< DistributedSystemRole >

Get a role.

Parameters

name	An identifier of an DistributedSystemRole

Returns

A shared pointer of specialized role

6.29.2.6 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in DistributedSystemArrayMediator, DistributedClientArrayMediator, and DistributedClient↩Array.

The documentation for this class was generated from the following file:

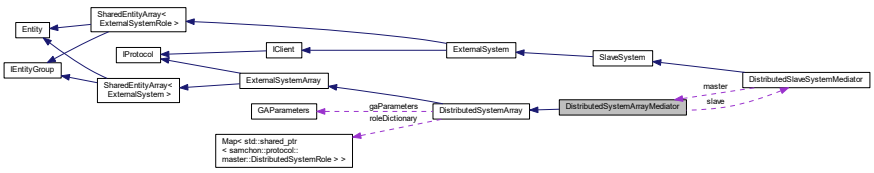
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSystemArray.hpp

6.30 DistributedSystemArrayMediator Class Reference

A mediator of distributed system between master and slaves.

```
#include <DistributedSystemArrayMediator.hpp>
```

Collaboration diagram for DistributedSystemArrayMediator:



Public Member Functions

- DistributedSystemArrayMediator ()

  Default Constructor.

- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.

- virtual void start () override

  Start interaction.

- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Protected Member Functions

- virtual auto createSlave () const -> DistributedSlaveSystemMediator *=0

  Factory method of slave system for mediation.

Protected Attributes

- DistributedSlaveSystemMediator * slave

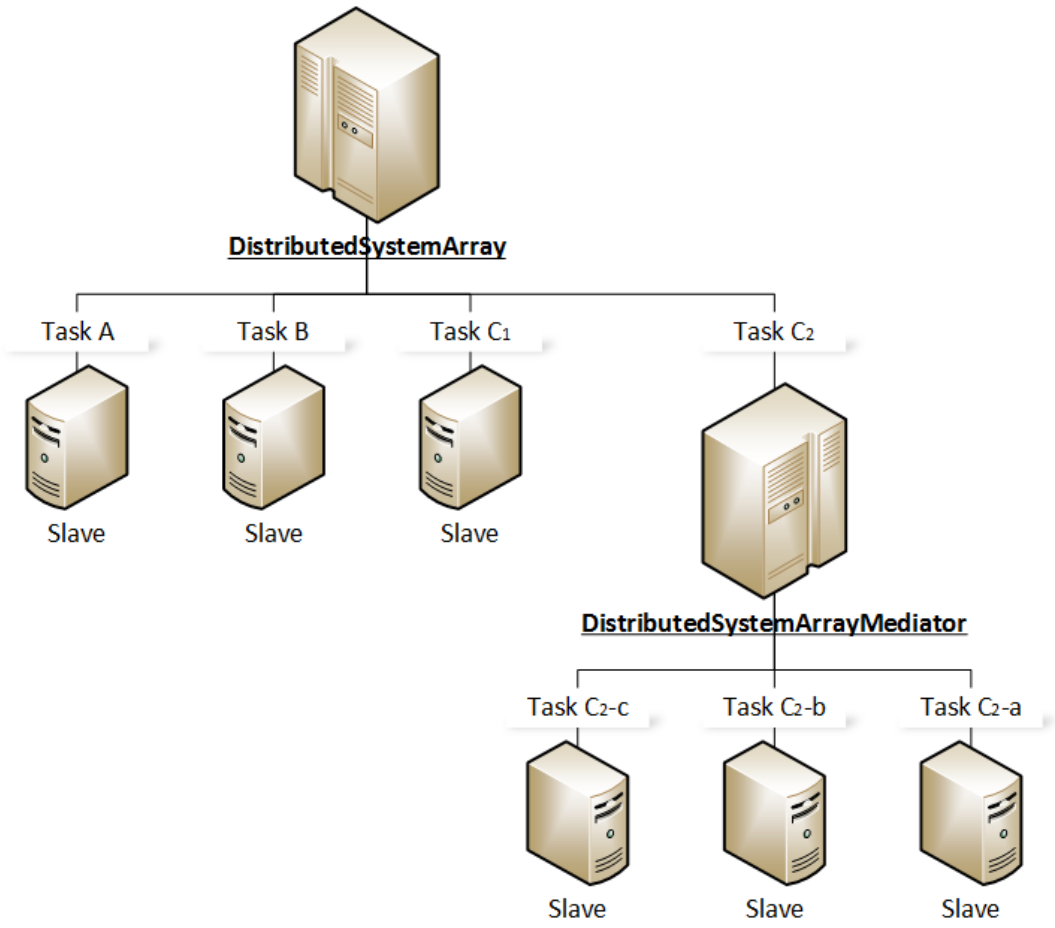
  A slave system for mediation.

6.30.1 Detailed Description

A mediator of distributed system between master and slaves.

DistributedSystemArrayMediator class is used to realizing a mediator system of tree-structured distributed processing system. DistributedSystemArrayMediator intermediates between origin-master(DistributedSystemArray) and its slaves(slave::DistributedSystem).

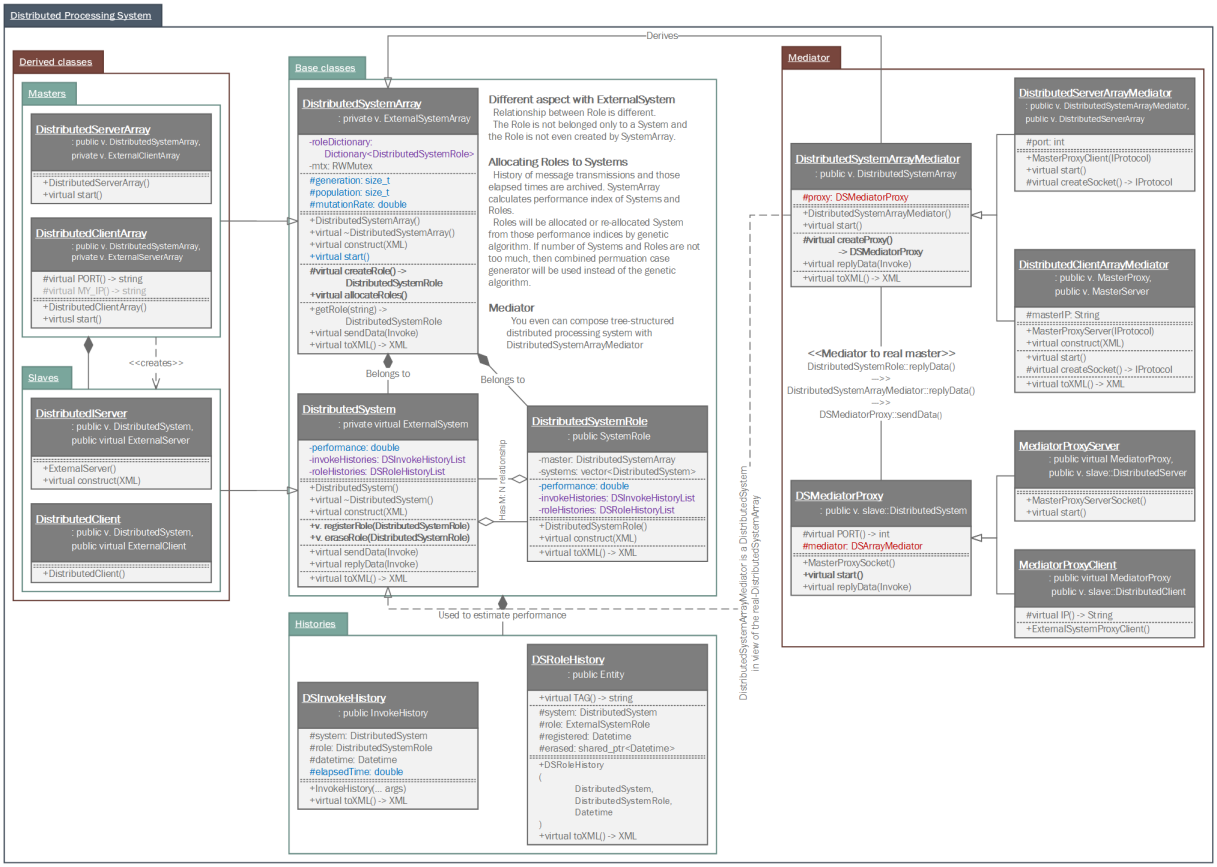
DistributedSystemArrayMediator is a master(DistributedSystemArray) within the framework of slaves systems (slave::DistributedSystem objects) and also can be a slave system (slave::DistributedSystem) within framework of its master(DistributedSystemArray) by having DistributedSlaveSystemMediator.



[Inherited]

DistributedSystemArray class is an ExternalSystemArray and an abstract class containing and managing distributed system drivers within framework of master. The DistributedSystemArray class allocates DistributedSystemRole objects to DistributedSystem objects for optimization.

Unlike the ExternalSystemRole class which is dependent on an ExternalSystem, DistributedSystemRole is not dependent on a DistributedSystem and even can be allocated to multiple DistributedSystem objects. Allocation of DistributedSystemRole(s) to DistributedSystem(s) is determined by estimation of performance index pursuing best distribution. If number of DistributedSystem and DistributedSystemRole objects is not a lot, optimize by combined permuation algorithm. Else the objects are a lot, the allocation will be optimized by genetic algorithm.



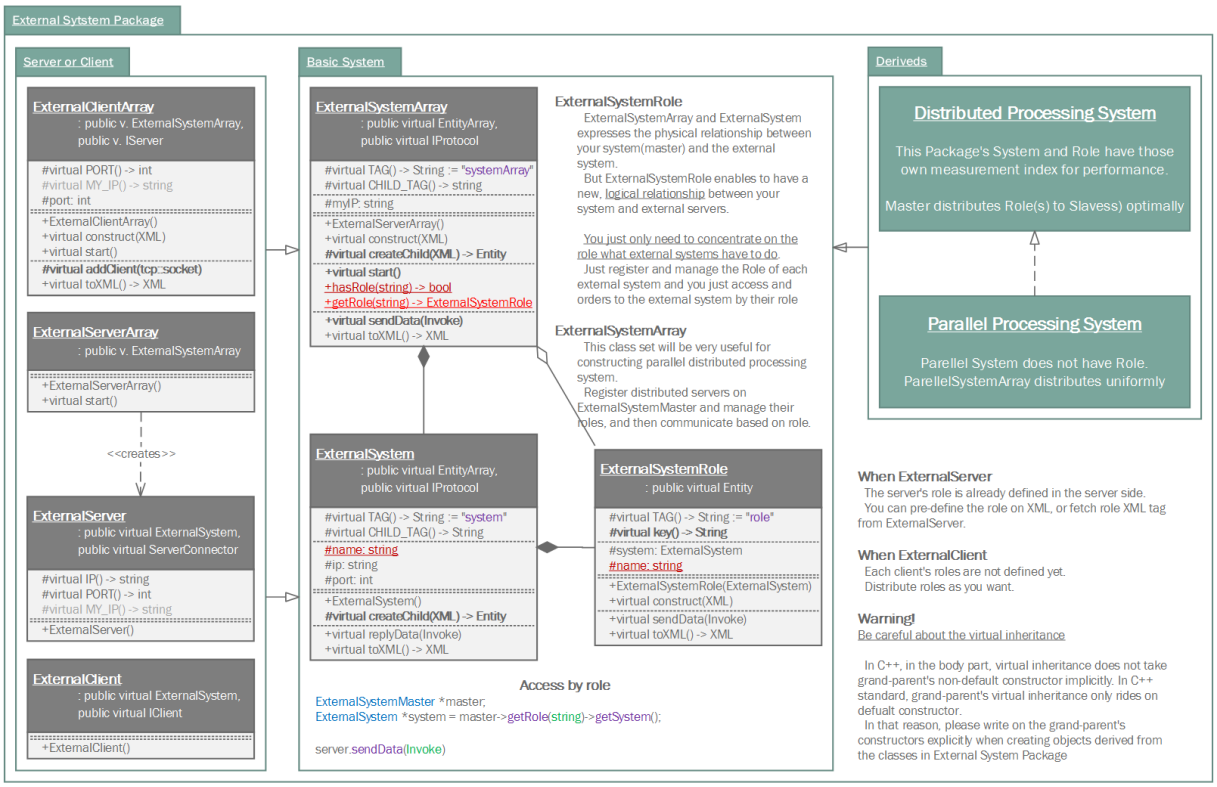
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.30.2 Member Function Documentation

6.30.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from DistributedSystemArray.

Reimplemented in DistributedClientArrayMediator.

6.30.2.2 virtual auto createSlave ( ) const -> DistributedSlaveSystemMediator * [protected], [pure virtual]

Factory method of slave system for mediation.

Exceptions

invalid_parameter	If DistributedServerArrayMediator, slave must be a type of DistributedSlave↩ ServerMediator

6.30.2.3 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Reimplemented from DistributedSystemArray.

Reimplemented in DistributedClientArrayMediator, and DistributedServerArrayMediator.

6.30.2.4 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from DistributedSystemArray.

Reimplemented in DistributedClientArrayMediator.

The documentation for this class was generated from the following file:

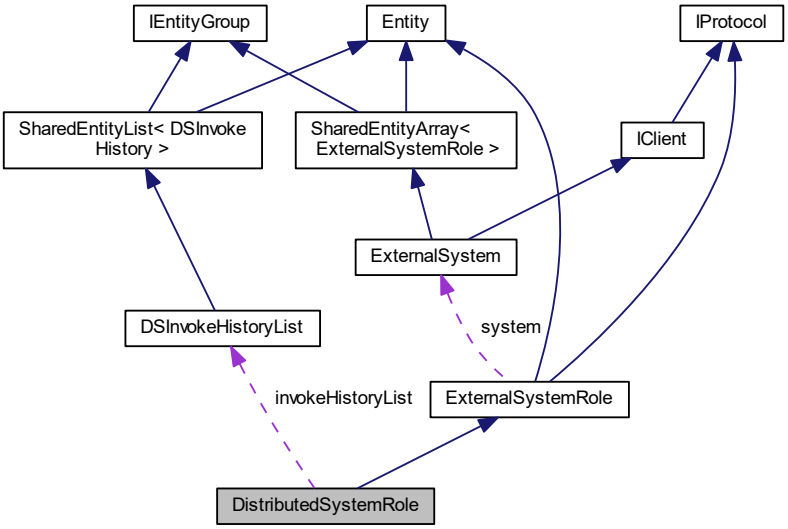
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSystemArray↩ Mediator.hpp

6.31 DistributedSystemRole Class Reference

A role of distributed processing system.

`#include <DistributedSystemRole.hpp>`

Collaboration diagram for DistributedSystemRole:



Public Member Functions

- DistributedSystemRole ()

  Default Constructor.
- auto getSystem () const -> ExternalSystem *=delete

  Get an external system -> depreciated.
- auto getPerformance () const -> double

  Get performance.
- auto getAllocationHistoryList () const -> DSRoleHistoryList *

  Get allocation histories.
- auto getInvokeHistoryList () const -> DSInvokeHistoryList *

  Get invoke histories.
- virtual void sendData (std::shared_ptr< Invoke >) override

  Send a message.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the Entity.

Protected Attributes

- std::set< DistributedSystem * > allocatedSystems

  Allocated systems of about the role, at now.
- double performance

  A required performance index.
- DSRoleHistoryList * allocationHistoryList

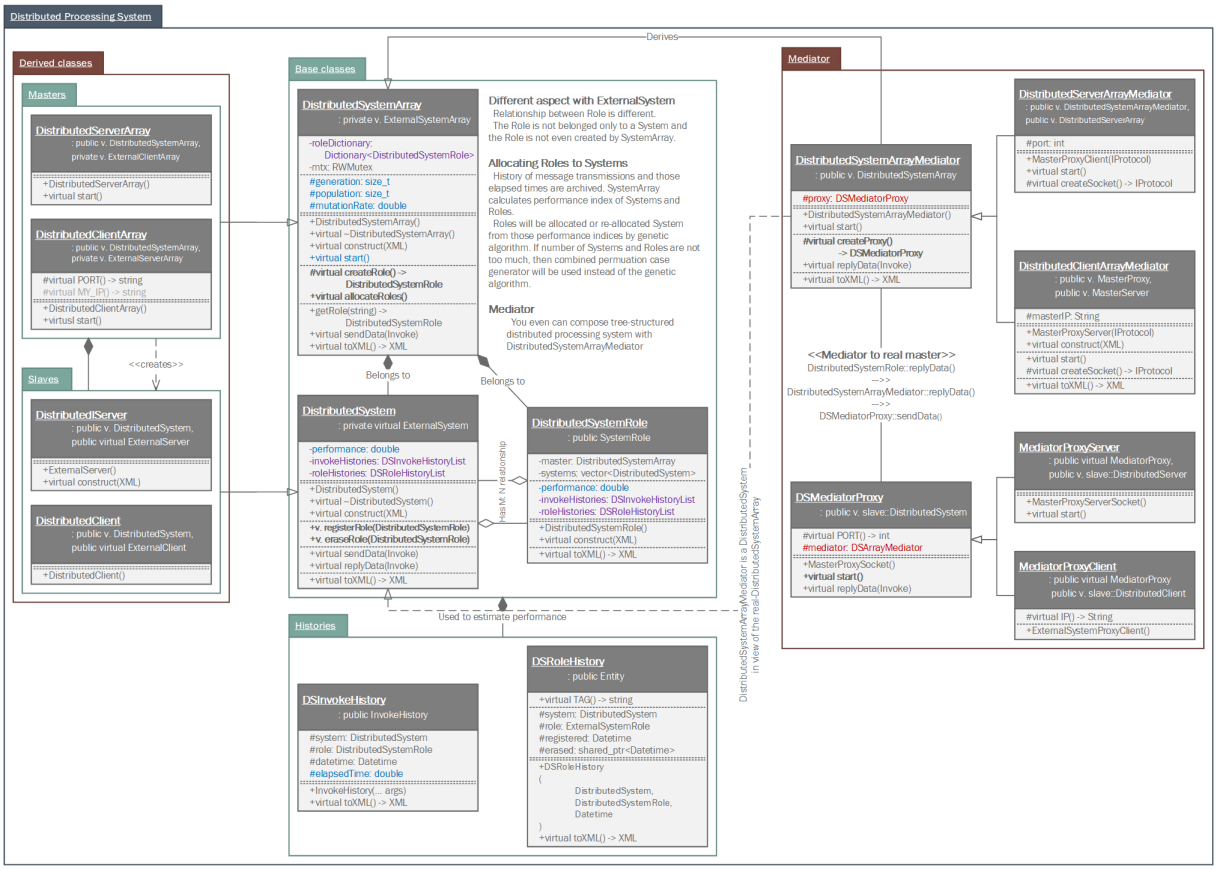
  A history log for archiving role allocation to a system.
- DSInvokeHistoryList * invokeHistoryList

  A history log for archiving elapsed time for each process.

6.31.1 Detailed Description

A role of distributed processing system.

DistributedSystemRole is an ExternalSystemRole allocates for distributed processing system(s). Unlike the ExternalSystemRole, DistributedSystemRole can belonged to multiple DistributedSystem(s).

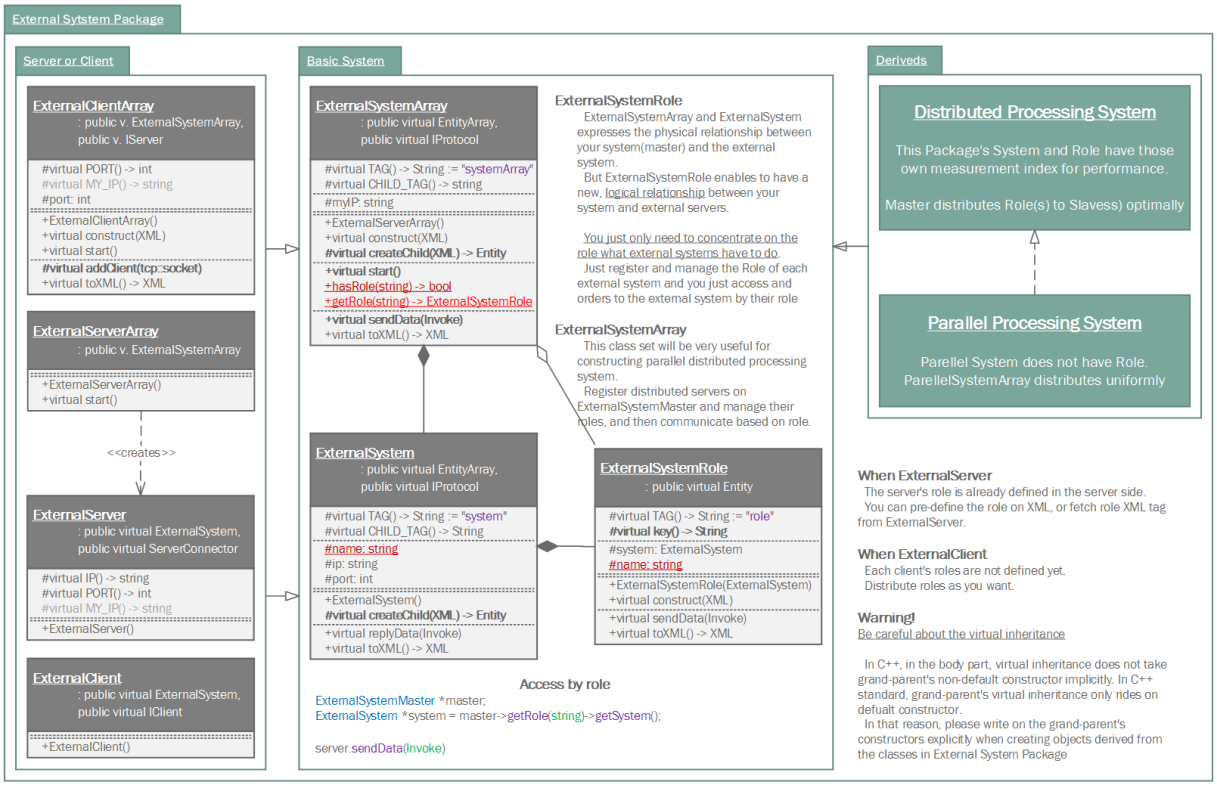


[Inherited]

ExternalSystemRole is a 'control' class groupping methods, handling Invoke messages interacting with an external system that the ExternalSystemRole is belonged to, by a subject or unit of a module.

ExternalSystemRole can be a "logical proxy" for an ExternalSystem which is containing the ExternalSystem↩ Role. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystem↩ Role acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.



See also

samchon::protocol

Author

Jeongho Nam

6.31.2 Member Function Documentation

6.31.2.1 auto getSystem ( ) const -> ExternalSystem *=delete [delete]

Get an external system -> depreciated.

Unlike ExternalSysemRole, DistributedSystemRole can be belonged to multiple DistributedSystem(s). By that reason, the getSystem(), a getter of a single ExternalSystem is depreciated.

6.31.2.2 virtual void sendData (std::shared_ptr< Invoke >) [override], [virtual]

Send a message.

Sends Invoke message to a network system or shifts the responsibility to related chain.

Parameters

invoke	An Invoke message to send to a network system

Reimplemented from ExternalSystemRole.

6.31.2.3 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ExternalSystemRole.

The documentation for this class was generated from the following file:

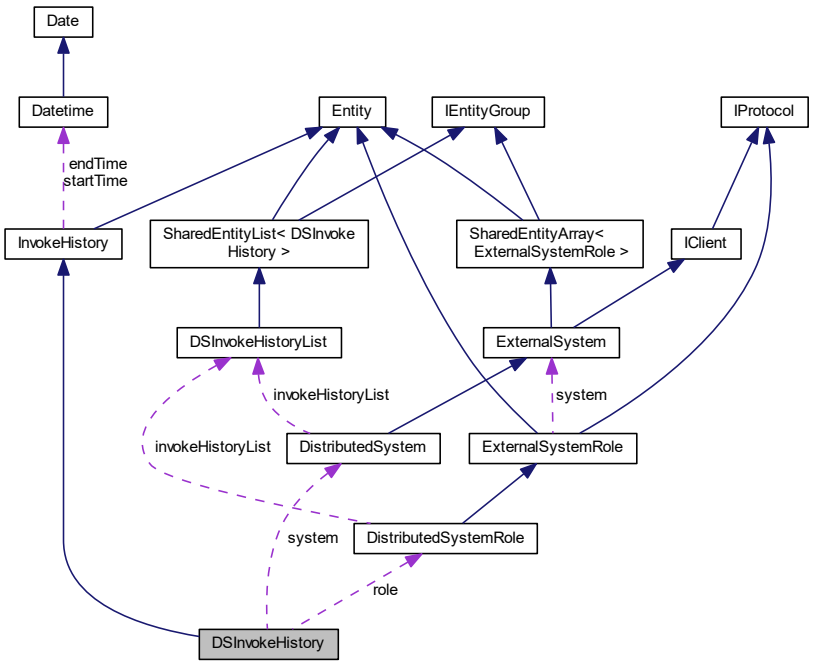
– D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DistributedSystemRole.hpp

6.32 DSInvokeHistory Class Reference

A reported history of an Invoke message.

`#include <DSInvokeHistory.hpp>`

Collaboration diagram for DSInvokeHistory:



Public Member Functions

- DSInvokeHistory (DistributedSystem *, DistributedSystemRole *)

  Construct from a system and role.

- auto getSystem () const -> DistributedSystem *

  Get source system.

- auto getRole () const -> DistributedSystemRole *

  Gets source role.

Protected Attributes

- DistributedSystem * system

  Source system.

- DistributedSystemRole * role

  Source role.

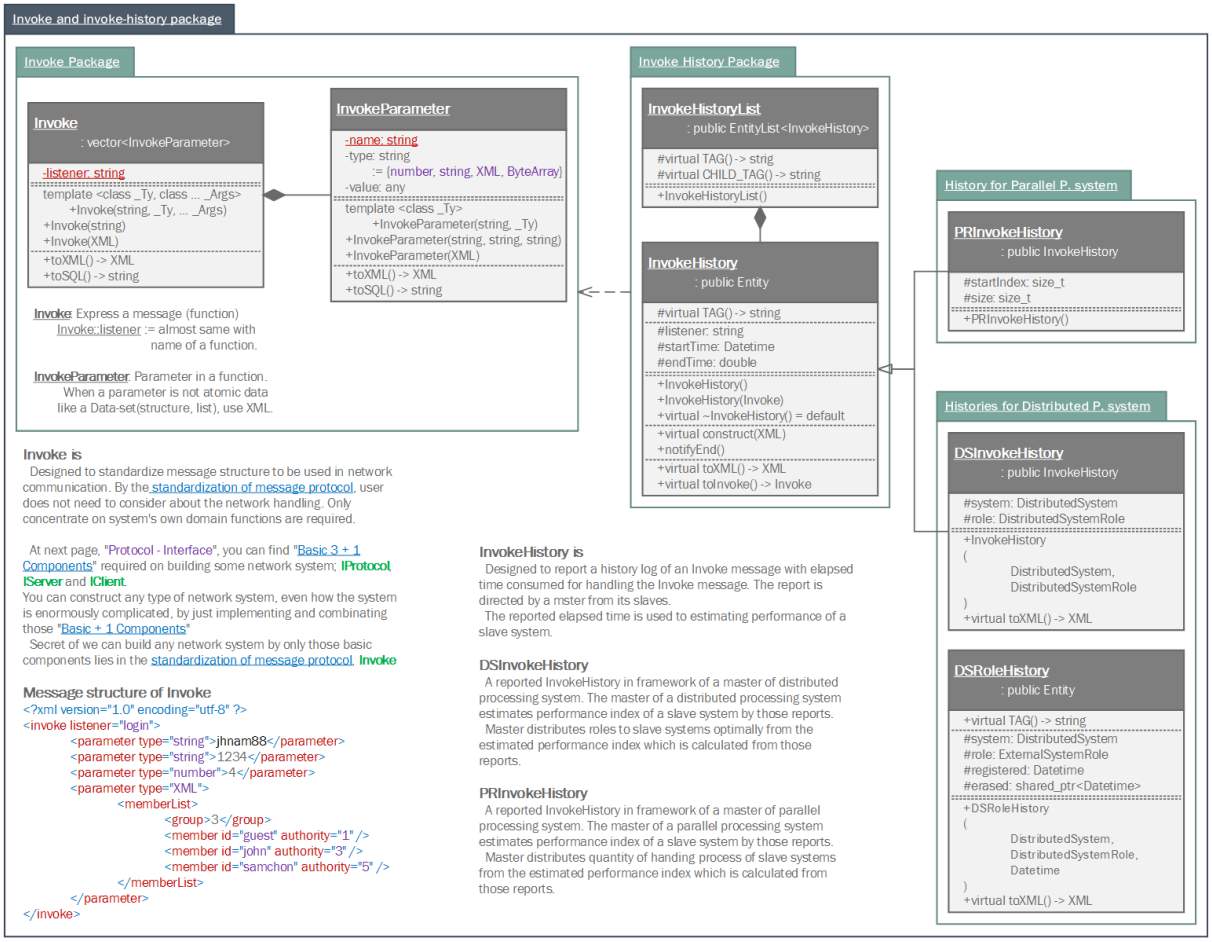
6.32.1 Detailed Description

A reported history of an Invoke message.

[Inherited]

InvokeHistory is a class for reporting history log of an Invoke message with elapsed time from a slave to its master.

With the elapsed time, consumed time for a process of handling the Invoke message, InvokeHistory is reported to the master. The master utilizies the elapsed time to estimating performances of each slave system. With the estimated performan index, master retrives the optimal solution of distributing processes.



See also

samchon::protocol
samchon::protocol::master
samchon::protocol::slave

Author

Jeongho Nam

6.32.2 Constructor & Destructor Documentation

6.32.2.1 DSInvokeHistory (DistributedSystem * *system*, DistributedSystemRole * *role*)

Construct from a system and role.

Parameters

system	a source system
role	a source role

The documentation for this class was generated from the following files:

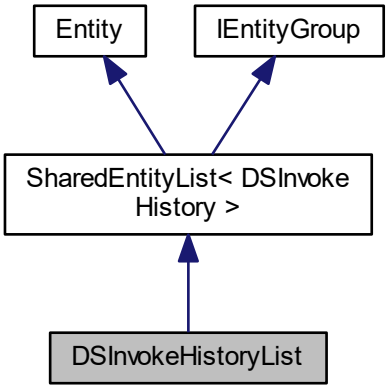
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DSInvokeHistory.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DSInvokeHistory.cpp

6.33 DSInvokeHistoryList Class Reference

A list of DSInvokeHistory.

```
#include <DSInvokeHistoryList.hpp>
```

Collaboration diagram for DSInvokeHistoryList:



Public Member Functions

- DSInvokeHistoryList ()
  Default Constructor.

Protected Member Functions

- virtual auto createChild (std::shared_ptr< library::XML >) -> DSInvokeHistory *override
  Factory method of a child Entity.

6.33.1 Detailed Description

A list of DSInvokeHistory.

Author

Jeongho Nam

6.33.2 Member Function Documentation

6.33.2.1 virtual auto createChild (std::shared_ptr< library::XML >) -> DSInvokeHistory *override [override], [protected], [virtual]

Factory method of a child Entity.

EntityGroup::createChild() is a factory method creating a new child Entity which is belonged to the Entity↩ Group. This method is called by EntityGroup::construct(). The children construction methods Entity↩ ::construct() will be called by abstract method of the EntityGroup::construct().

Returns

A new child Entity belongs to EntityGroup.

Implements EntityGroup< _Container, _ETy, _Ty >.

The documentation for this class was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/DSInvokeHistoryList.hpp

6.34 Entity Class Reference

An entity, a standard data class.

```
#include <Entity.hpp>
```

Public Member Functions

- virtual auto TAG () const -> std::string=0

  A tag name when represented by XML.

- Entity ()

  Default Constructor.

- virtual void construct (std::shared_ptr< library::XML >)=0

  Construct data of the Entity from an XML object.

- virtual auto key () const -> std::string

  Get a key that can identify the Entity uniquely.

- virtual auto toXML () const -> std::shared_ptr< library::XML >

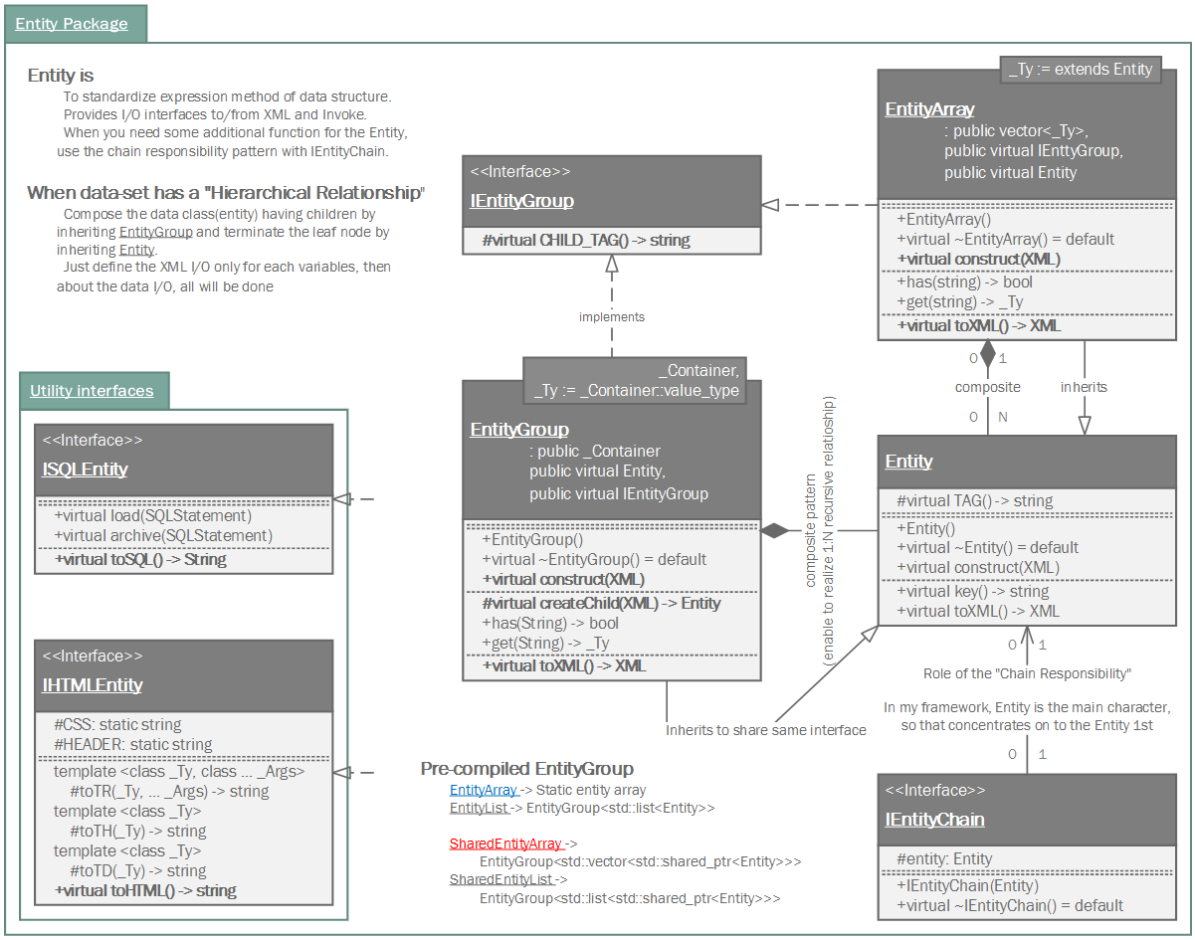
  Get an XML object represents the Entity.

6.34.1 Detailed Description

An entity, a standard data class.

Entity is a class for standardization of expression method using on network I/O by XML. If Invoke is a standard message protocol of Samchon Framework which must be kept, Entity is a recommended semi-protocol of message for expressing a data class. Following the semi-protocol Entity is not imposed but encouraged.

As we could get advantages from standardization of message for network I/O with Invoke, we can get additional advantage from standardizing expression method of data class with Entity. We do not need to know a part of network communication. Thus, with the Entity, we can only concentrate on entity's own logics and relationships between another entities. Entity does not need to how network communications are being done.



Example source

```
#include <iostream>

#include <samchon/protocol/Entity.hpp>
#include <samchon/protocol/IHTMLEntity.hpp>
#include <samchon/protocol/SharedEntityArray.hpp>

#include <samchon/library/XML.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;

class Member
    : public Entity, public virtual IHTMLEntity
{
protected:
    typedef Entity super;

    string id;
    string name;
    int age;
    int grade;

public:
    /* ----------------------------------------------------------------------
        CONSTRUCTORS
```

```
    ------------------------------------------------------------------- */
    Member()
        : super(), IHTMLEntity()
    {
    };
    Member(const string &id, const string &name, int age, int grade)
        : super(), IHTMLEntity()
    {
        this->id = id;
        this->name = name;
        this->age = age;
        this->grade = grade;
    };
    virtual ~Member() = default;

    virtual void construct(shared_ptr<XML> xml) override
    {
        this->id = xml->getProperty("id");
        this->name = xml->getProperty("name");
        this->age = xml->getProperty<int>("age");
        this->grade = xml->getProperty<int>("grade");
    };

    /* ------------------------------------------------------------------
        GETTERSs
    ------------------------------------------------------------------- */
    virtual auto key() const -> std::string override
    {
        return this->id;
    };

    /* ------------------------------------------------------------------
        XML EXPORTERS
    ------------------------------------------------------------------- */
    virtual auto TAG() const -> string override
    {
        return "member";
    };
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("id", id);
        xml->setProperty("name", name);
        xml->setProperty("age", age);
        xml->setProperty("grade", grade);

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        return toTR(id, name, age, grade);
    };
};

class MemberArray
    : public SharedEntityArray<Member>,
    public virtual IHTMLEntity
{
protected:
    typedef SharedEntityArray<Member> super;

    string application;
    int department;
    Member *chief;

public:
    /* ------------------------------------------------------------------
        CONSTRUCTORS
    ------------------------------------------------------------------- */
    MemberArray()
        : super(), IHTMLEntity()
    {
        this->chief = nullptr;
    };
    virtual ~MemberArray() = default;

    // You don't need to consider children(Member) objects
    // Just concentrate on constructing MemberArray's own member variables
    virtual void construct(shared_ptr<XML> xml) override
    {
        super::construct(xml);

        this->application = xml->getProperty("application");
        this->department = xml->getProperty<int>("department");

        if(xml->hasProperty("chief") == true && this->has( xml->getProperty("chief") ) == true)
            this->chief = this->get( xml->getProperty("cheif") ).get();
```

```
    };

protected:
    //FACTORY METHOD FOR MEMBER
    virtual auto createChild(shared_ptr<XML> = nullptr) -> Member* override
    {
        return new Member();
    };

    /* ---------------------------------------------------------------------
        XML EXPORTERS
    --------------------------------------------------------------------- */
public:
    virtual auto TAG() const -> string override
    {
        return "memberArray";
    };
    virtual auto CHILD_TAG() const -> string override
    {
        return "member";
    };

    // You don't need to consider children(Member) objects
    // Just concentrate on expressing MemberArray's own member variables
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("application", application);
        xml->setProperty("department", department);

        if(chief != nullptr)
            xml->setProperty("cheif", chief->key());

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        string html = "<table>\n";
        html += toTH("ID", "Name", "Age", "Grade") + "\n";

        for (size_t i = 0; i < 2; i++)
            html += at(i)->toHTML() + "\n";

        html += "</table>";
        return move(html);
    };
};

void main()
{
    string str = string("") +
        "<memberArray application='framework' department='7' cheif='samchon'>\n" +
        "   <member id='samchon' name='Jeongho Nam' age='27' grade='5' />" +
        "   <member id='submaster' name='No Name' age='100' grade='4' />" +
        "   <member id='john' name='John Doe' age='33' grade='2' />" +
        "   <member id='bad_man' name='Bad Man' age='44' grade='-1' />" +
        "   <member id='guest' name='Guest' age='0' grade='0' />" +
        "</memberArray>";
    shared_ptr<XML> xml(new XML(str));

    MemberArray memberArray;
    memberArray.construct(xml);

    memberArray.emplace_back(new Member("freshman", "A fresh man", 20, 2));
    memberArray.emplace_back(new Member("senior", "A senior", 70, 2));

    cout << memberArray.toXML()->toString() << endl << endl;
    cout << memberArray.toHTML() << endl;
    system("pause");
}
```

Result of the example

```
D:\OneDrive\Project\Samchon\framework\cpp\Debug\entity.exe
<memberArray application="framework" department="7">
        <member age="27" grade="5" id="samchon" name="Jeongho" />
        <member age="100" grade="4" id="submaster" name="No" />
        <member age="33" grade="2" id="john" name="John" />
        <member age="44" grade="-1" id="bad_man" name="Bad" />
        <member age="0" grade="0" id="guest" name="Guest" />
        <member age="20" grade="2" id="freshman" name="A fresh man" />
        <member age="70" grade="2" id="senior" name="A senior" />
</memberArray>

<table>
        <th>
                <td>ID</td>
                <td>Name</td>
                <td>Age</td>
                <td>Grade</td>
        </th>
        <tr>
                <td>samchon</td>
                <td>Jeongho</td>
                <td>27</td>
                <td>5</td>
        </tr>
        <tr>
                <td>submaster</td>
                <td>No</td>
                <td>100</td>
                <td>4</td>
        </tr>
</table>
Press any key to continue . . .
```

Note

I say repeatedly. Expression method of Entity is recommended, but not imposed. It's a semi protocol for network I/O but not a essential protocol must be kept. The expression method of Entity, using on network I/O, is expressed by XML string.

If your own network system has a critical performance issue on communication data class, it would be better to using binary communication (with ByteArray or boost::serialization). Don't worry about the problem! Invoke also provides methods for binary data (ByteArray).

See also

protocol

Author

Jeongho Nam

6.34.2 Member Function Documentation

6.34.2.1 virtual auto TAG ( ) const -> std::string [pure virtual]

A tag name when represented by XML.

Returns

A tag name

Implemented in InvokeParameter, InvokeHistory, Invoke, NTCriteria, ExternalSystemArray, ExternalSystem, ExternalSystemRole, NTParameter, NTParameterDetermined, SystemRole, DistributedSlaveSystemMediator, ParallelSlaveSystemMediator, ChatRoom, NTSide, InvokeHistoryArray, NTParameterArray, ChatMessage, and FTInstance.

6.34.2.2 void construct (std::shared_ptr< library::XML >) [pure virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implemented in InvokeParameter, DistributedSystem, NTCriteria, NTFile, InvokeHistory, EntityGroup< _Container, _ETy, _Ty >, EntityGroup< _Container, std::unique_ptr< Entity > >, EntityGroup< _Container, std::shared_ptr< Entity > >, EntityGroup< _Container, Entity * >, EntityList< _Ty >, ExternalSystem, ExternalSystemRole, ParallelSystem, EntityArray< _Ty >, Invoke, DistributedSystemArray, ChatMessage, INTExplore, NTParameterDetermined, DistributedSystemArrayMediator, ParallelSystemArrayMediator, PRInvokeHistory, NTParameter, SystemRole, NTSide, ExternalClientArray, ExternalServer, DistributedClientArrayMediator, ParallelClientArrayMediator, DistributedClientArray, ParallelServer, and DistributedServer.

6.34.2.3 auto key ( ) const -> std::string [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented in InvokeParameter, InvokeHistory, ExternalSystem, ExternalSystemRole, FTInstance, NTParameter, NTParameterDetermined, and SystemRole.

6.34.2.4 auto toXML ( ) const -> std::shared_ptr<library::XML> [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented in InvokeParameter, EntityGroup< _Container, _ETy, _Ty >, EntityGroup< _Container, std::unique_ptr< Entity > >, EntityGroup< _Container, std::shared_ptr< Entity > >, EntityGroup< _Container, Entity * >, InvokeHistory, Invoke, DistributedSystem, NTCriteria, NTFile, ExternalSystem, ExternalSystemRole, DistributedSystemArray, EntityList< _Ty >, DistributedSystemRole, EntityArray< _Ty >, ParallelSystem, ChatRoom, FTInstance, INTExplore, NTParameter, NTParameterDetermined, PRInvokeHistory, ChatMessage, DistributedSystemArrayMediator, ParallelSystemArrayMediator, SystemRole, ExternalClientArray, ExternalServer, FTFolder, NTSide, FTFile, DistributedClientArrayMediator, ParallelClientArrayMediator, DistributedClientArray, ParallelServer, and DistributedServer.

The documentation for this class was generated from the following files:

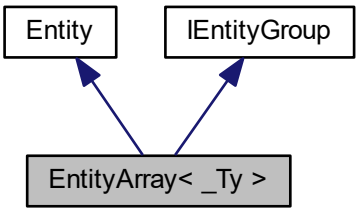
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/Entity.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/Entity.cpp

6.35 EntityArray< _Ty > Class Template Reference

An Entity and a static array containing Entity objects.

```
#include <EntityArray.hpp>
```

Collaboration diagram for EntityArray< _Ty >:



Public Member Functions

- EntityArray ()
  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML > xml) override
  Construct data of the Entity from an XML object.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the EntityArray.

Additional Inherited Members

6.35.1 Detailed Description

template<typename _Ty>class samchon::protocol::EntityArray< _Ty >

An Entity and a static array containing Entity objects.

Template Parameters

_Ty	A type of children Entity. Must be a class derived from an Entity.

EntityArray is a static array containing children objects derived from an Entity class.

The EntityArray has advantages of performance like fast iteration or fast access because the EntityArray is serialized. However, as EntityArray is serialized, allocated memory space for each child is fixed and cannot contain derived a entity class from the child entity type.

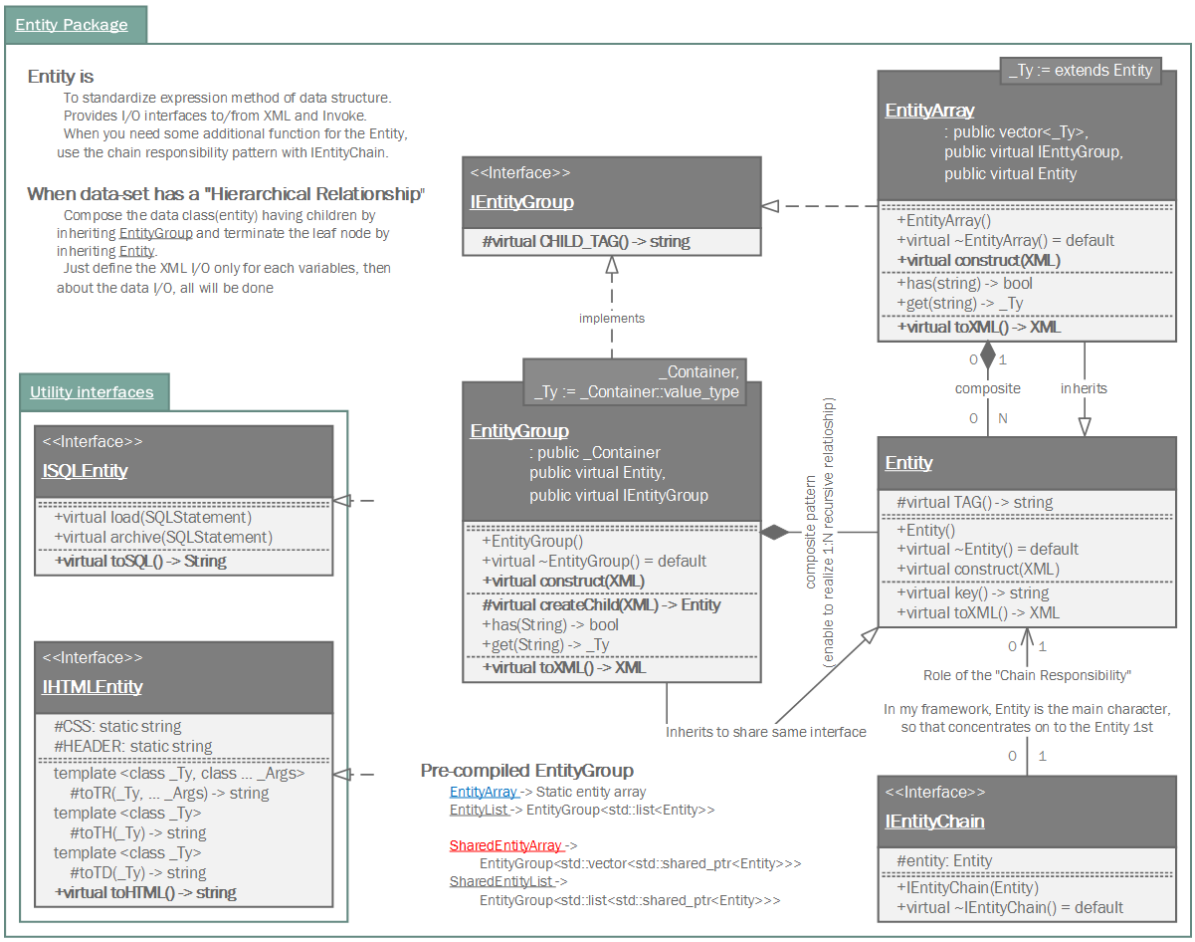
- EntityArray<Animal> is specified
- Dog is extended from the Animal
- EntityArray<Animal> cannot contain Dog. You can try insertion of the Dog class into the EntityArray, but only memory size of Animal is allocated, so that derived member variables and methods are all truncated.

Thus, it's suitable for when fast accessment and iteration like an entity class containing mathmatical or statistical data and methods which doesn't seemed to be inherited.

[Inherited]

Entity is a class for standardization of expression method using on network I/O by XML. If Invoke is a standard message protocol of Samchon Framework which must be kept, Entity is a recommended semi-protocol of message for expressing a data class. Following the semi-protocol Entity is not imposed but encouraged.

As we could get advantages from standardization of message for network I/O with Invoke, we can get additional advantage from standardizing expression method of data class with Entity. We do not need to know a part of network communication. Thus, with the Entity, we can only concentrate on entity's own logics and relationships between another entities. Entity does not need to how network communications are being done.



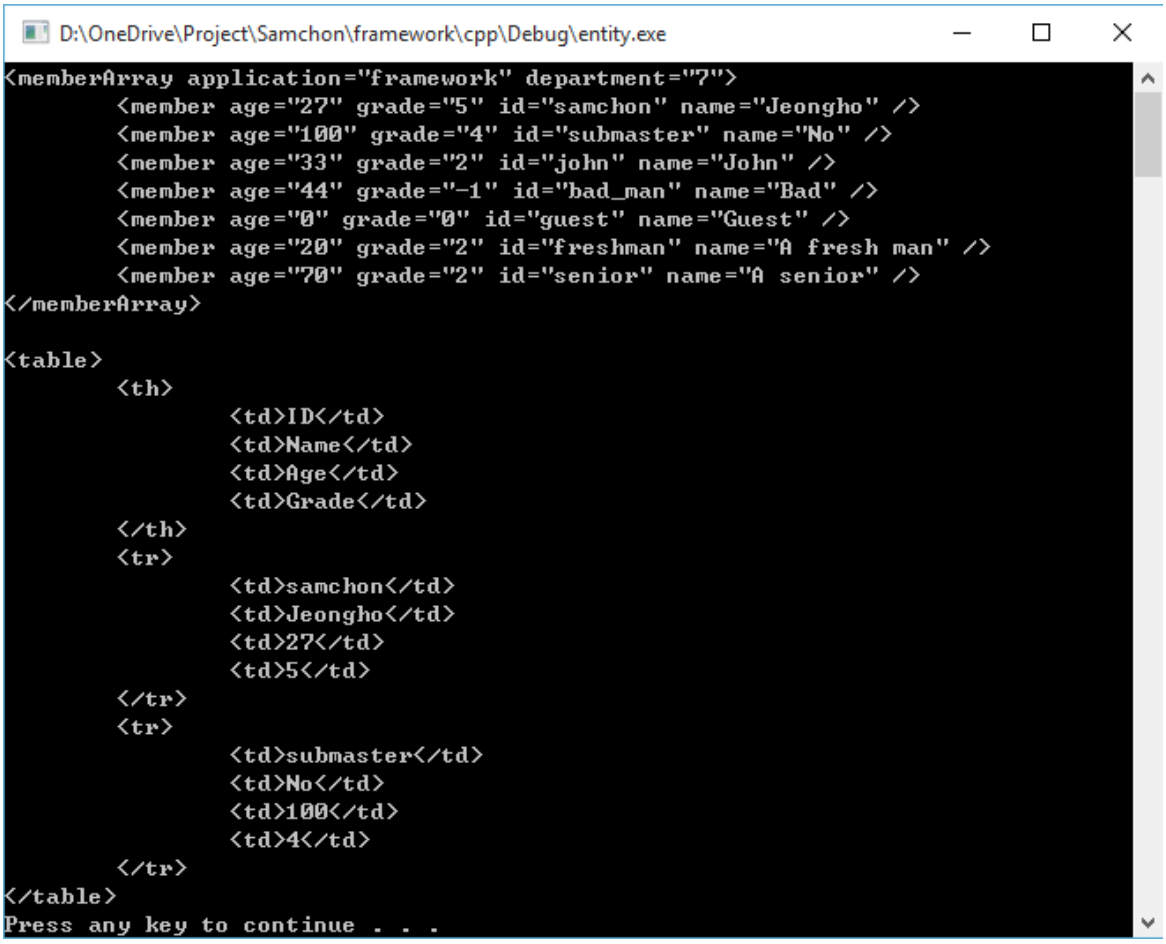
Example source

```
#include <iostream>

#include <samchon/protocol/Entity.hpp>
#include <samchon/protocol/IHTMLEntity.hpp>
#include <samchon/protocol/SharedEntityArray.hpp>

#include <samchon/library/XML.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;

class Member
    : public Entity, public virtual IHTMLEntity
{
protected:
    typedef Entity super;

    string id;
    string name;
    int age;
    int grade;

public:
    /* ----------------------------------------------------------------------
        CONSTRUCTORS
    ---------------------------------------------------------------------- */
    Member()
        : super(), IHTMLEntity()
    {
    };
    Member(const string &id, const string &name, int age, int grade)
        : super(), IHTMLEntity()
    {
        this->id = id;
        this->name = name;
        this->age = age;
        this->grade = grade;
    };
    virtual ~Member() = default;

    virtual void construct(shared_ptr<XML> xml) override
    {
        this->id = xml->getProperty("id");
        this->name = xml->getProperty("name");
        this->age = xml->getProperty<int>("age");
        this->grade = xml->getProperty<int>("grade");
    };

    /* ----------------------------------------------------------------------
        GETTERSs
    ---------------------------------------------------------------------- */
    virtual auto key() const -> std::string override
    {
        return this->id;
    };

    /* ----------------------------------------------------------------------
        XML EXPORTERS
    ---------------------------------------------------------------------- */
    virtual auto TAG() const -> string override
    {
        return "member";
    };
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("id", id);
        xml->setProperty("name", name);
        xml->setProperty("age", age);
        xml->setProperty("grade", grade);
```

```

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        return toTR(id, name, age, grade);
    };
};

class MemberArray
    : public SharedEntityArray<Member>,
    public virtual IHTMLEntity
{
protected:
    typedef SharedEntityArray<Member> super;

    string application;
    int department;
    Member *chief;

public:
    /* ----------------------------------------------------------------------
        CONSTRUCTORS
    ---------------------------------------------------------------------- */
    MemberArray()
        : super(), IHTMLEntity()
    {
        this->chief = nullptr;
    };
    virtual ~MemberArray() = default;

    // You don't need to consider children(Member) objects
    // Just concentrate on constructing MemberArray's own member variables
    virtual void construct(shared_ptr<XML> xml) override
    {
        super::construct(xml);

        this->application = xml->getProperty("application");
        this->department = xml->getProperty<int>("department");

        if(xml->hasProperty("chief") == true && this->has( xml->getProperty("chief") ) == true)
            this->chief = this->get( xml->getProperty("cheif") ).get();
    };

protected:
    //FACTORY METHOD FOR MEMBER
    virtual auto createChild(shared_ptr<XML> = nullptr) -> Member* override
    {
        return new Member();
    };

    /* ----------------------------------------------------------------------
        XML EXPORTERS
    ---------------------------------------------------------------------- */
public:
    virtual auto TAG() const -> string override
    {
        return "memberArray";
    };
    virtual auto CHILD_TAG() const -> string override
    {
        return "member";
    };

    // You don't need to consider children(Member) objects
    // Just concentrate on expressing MemberArray's own member variables
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("application", application);
        xml->setProperty("department", department);

        if(chief != nullptr)
            xml->setProperty("cheif", chief->key());

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        string html = "<table>\n";
        html += toTH("ID", "Name", "Age", "Grade") + "\n";

        for (size_t i = 0; i < 2; i++)
            html += at(i)->toHTML() + "\n";

        html += "</table>";
        return move(html);
```

```
    };
};

void main()
{
    string str = string("") +
        "<memberArray application='framework' department='7' cheif='samchon'>\n" +
        "   <member id='samchon' name='Jeongho Nam' age='27' grade='5' />" +
        "   <member id='submaster' name='No Name' age='100' grade='4' />" +
        "   <member id='john' name='John Doe' age='33' grade='2' />" +
        "   <member id='bad_man' name='Bad Man' age='44' grade='-1' />" +
        "   <member id='guest' name='Guest' age='0' grade='0' />" +
        "</memberArray>";
    shared_ptr<XML> xml(new XML(str));

    MemberArray memberArray;
    memberArray.construct(xml);

    memberArray.emplace_back(new Member("freshman", "A fresh man", 20, 2));
    memberArray.emplace_back(new Member("senior", "A senior", 70, 2));

    cout << memberArray.toXML()->toString() << endl << endl;
    cout << memberArray.toHTML() << endl;
    system("pause");
}
```

Result of the example



Note

I say repeatedly. Expression method of Entity is recommended, but not imposed. It's a semi protocol for network I/O but not a essential protocol must be kept. The expression method of Entity, using on network I/O, is expressed by XML string.

If your own network system has a critical performance issue on communication data class, it would be better to using binary communication (with ByteArray or boost::serialization). Don't worry about the problem! Invoke also provides methods for binary data (ByteArray).

See also

protocol

Author

Jeongho Nam

Warning

If a data structure has recursive and hierarchical relationship, never use the EntityArray. EntityArray's memory allocation is static. The recursive relationship causes infinite memory allocation. It must generates stack overflow.

Use EntityGroup instead.

- class NTCriteria : public EntityArray<NTCriteria> causes a stack overflow.
- class NTCriteria : public SharedEntityArray<NTCriteria> doesn't cause the stack overflow.

6.35.2 Member Function Documentation

6.35.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [inline], [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityArray's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityArray::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityArray::toXML().

Constructs only data of EntityArray's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

6.35.2.2 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [inline], [override], [virtual]

Get an XML object represents the EntityArray.

Archives the EntityArray's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityArray::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityArray::toXML().

Archives only data of EntityArray's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

The documentation for this class was generated from the following file:

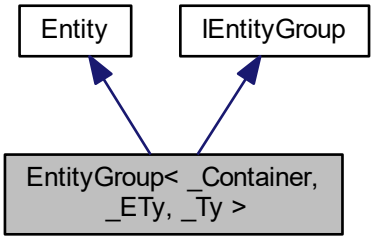
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/EntityArray.hpp

6.36 EntityGroup< _Container, _ETy, _Ty > Class Template Reference

An Entity and a container of children Entity objects.

`#include <EntityGroup.hpp>`

Collaboration diagram for EntityGroup< _Container, _ETy, _Ty >:



Public Member Functions

- EntityGroup ()
  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML > xml)
  Construct data of the Entity from an XML object.
- auto has (const std::string &key) const -> bool
  Indicates whether a container has an object having the specified identifier.

- auto get (const std::string &key) -> value_type &

  Access the element by specified identifier(key).

- auto get (const std::string &key) const -> const value_type &

  Access the const element by specified identifier(key).

- virtual auto toXML () const -> std::shared_ptr< library::XML >

  Get an XML object represents the EntityGroup.

Protected Member Functions

- virtual auto createChild (std::shared_ptr< library::XML >) -> entity_type *=0

  Factory method of a child Entity.

6.36.1 Detailed Description

template<typename _Container, typename _ETy = Entity, typename _Ty = _Container::value_type>class samchon::protocol::EntityGroup< _Container, _ETy, _Ty >

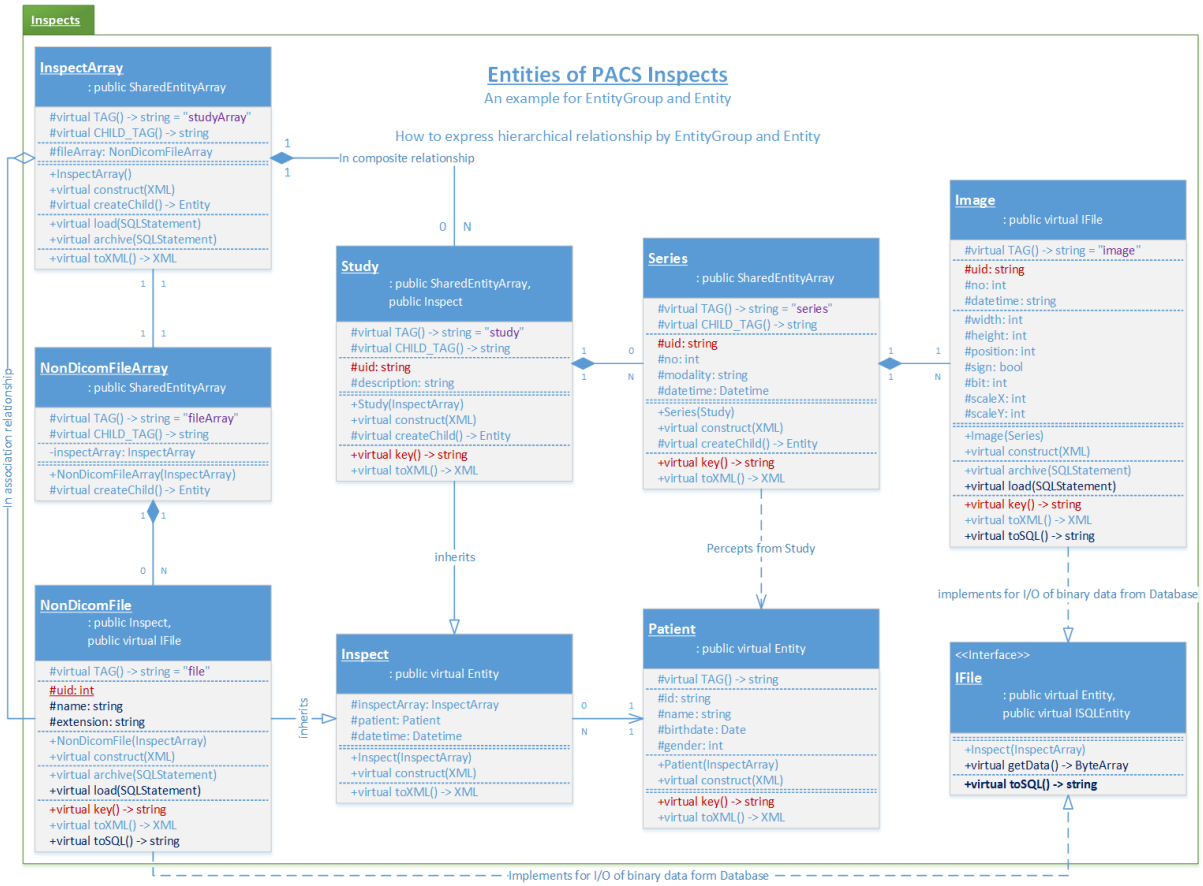
An Entity and a container of children Entity objects.

Template Parameters

_Container	A type of container containing children entity objects.
_Ety	A type of children entity. It must be a class derived from an Entity class, or Entity class itself.
_Ty	A type of children element of _Container. Using default template parameter is recommended.

EntityGroup is a template class for containinig children Entity objects, and also another type of an Entity, too. You can realize hierarchical relationship. Although some entities have complicated hierarchical relationship, you can deduct a optimal solution easily with EntityGroup and Entity.

If an entity has some subordinate entities of same type, they are in "Composite relationship". Make the entity to be EmntityGroup and subordinate entities to be children of the entity. When those relationships are continued, continue to create classes dervied from EntityGroup. When those relationshiop meets a terminal node, then make the terminal node to be an Entity.



EntityGroup is an Entity, and a container of children Entity objects at the same time. If children type, of a class derived from an EntityGroup, is itself, you can realize hierarchical and recursive relationship. The relationship is called as "Composite pattern".

- FTFolder extends FTInstance and SharedEntityArray<FTInstance>
- NTCriteria extends SharedEntityArray<NTCriteria>

Note

As a freelancer developer and architect I am, I even design DB I/O to follow the format representing Entity and EntityGroup by XML. Below T-SQL script also follows the standard format of expressing Entity with XML by procedure and "FOR XML AUTO" statement.

```
SET
  ANSI_NULLS,
  QUOTED_IDENTIFIER,
  CONCAT_NULL_YIELDS_NULL,
  ANSI_WARNINGS,
  ANSI_PADDING
ON;

USE OraQ;
IF OBJECT_ID('v_studyStatus') IS NOT NULL            DROP VIEW v_studyStatus;
IF OBJECT_ID('v_nonDicomFileStatus') IS NOT NULL     DROP VIEW v_nonDicomFileStatus;
IF OBJECT_ID('v_candidateStudy') IS NOT NULL         DROP VIEW v_candidateStudy;
IF OBJECT_ID('v_candidateFile') IS NOT NULL          DROP VIEW v_candidateFile;
IF OBJECT_ID('goCandidateInspectList') IS NOT NULL   DROP PROCEDURE goCandidateInspectList;
GO

CREATE VIEW v_studyStatus
AS
    WITH cte AS
    (
        SELECT S.studyUID uid, S.type type, ROW_NUMBER() OVER (PARTITION BY S.studyUID ORDER BY H.datetime
    DESC) no
```

```
        FROM History.history H, History.study S
        WHERE
            H.uid = S.uid AND type <> 1
    )
    SELECT
        A.uid,
        CASE WHEN B.type IS NULL THEN
            A.type
            ELSE 1
        END status
    FROM
    (
        SELECT uid, type
        FROM cte
        WHERE no = 1
    ) A LEFT OUTER JOIN
    (
        SELECT studyUID uid, 1 type
        FROM History.study
        GROUP BY studyUID
        HAVING MAX(type) = 1
    ) B ON A.uid = B.uid
GO
CREATE VIEW v_nonDicomFileStatus
AS
    WITH cte AS
    (
        SELECT
            F.fileUID uid, F.type type,
            ROW_NUMBER() OVER (PARTITION BY F.fileUID ORDER BY H.datetime DESC) no
        FROM History.history H, History.nonDicomFile F
        WHERE
            H.uid = F.uid AND type <> 1
    )
    SELECT
        A.uid,
        CASE WHEN B.type IS NULL THEN
            A.type
            ELSE 1
        END status
    FROM
    (
        SELECT uid, type
        FROM cte
        WHERE no = 1
    ) A LEFT OUTER JOIN
    (
        SELECT fileUID uid, 1 type
        FROM History.nonDicomFile
        GROUP BY fileUID
        HAVING MAX(type) = 1
    ) B ON A.uid = B.uid
GO
CREATE VIEW v_candidateStudy
AS
    SELECT DISTINCT
        P.id patientID,
            P.name patientName, P.gender patientGender,
            CAST(P.birthdate AS DATE) patientBirthdate,
        S.uid,
            S.description,
            CONVERT(VARCHAR, CAST(S.datetime AS DATETIME), 120) AS datetime,
            H.status
    FROM
        Hospital.patient P, Inspect.media M, Inspect.mediaStudyPair A,
        Inspect.study S, v_studyStatus H
    WHERE
        P.id = M.patientID AND M.uid = A.mediaUID AND A.studyUID = S.uid AND
        S.uid = H.uid
GO
CREATE VIEW v_candidateFile
AS
    SELECT DISTINCT
        P.id patientID,
            P.name patientName, P.gender patientGender,
            CAST(P.birthdate AS DATE) patientBirthdate,
        M.uid mediaUID,
            F.uid, F.name, F.extension,
             CONVERT(VARCHAR, CAST(F.datetime AS DATETIME), 120) AS datetime,
             H.status
     FROM
         Hospital.patient P, Inspect.media M,
         Inspect.nonDicomFile F, v_nonDicomFileStatus H
     WHERE
         P.id = M.patientID AND M.uid = F.mediaUID AND F.uid = H.uid
GO

```

```
CREATE PROCEDURE goCandidateInspectList
AS
    DECLARE @xml XML =
    (
        SELECT *
        FROM
        (
            SELECT N'candidate' service
        ) inspect
        FOR XML AUTO
    )
    DECLARE @studyList XML =
    (
        SELECT study.*, series.*, image.*
        FROM
            v_candidateStudy study
                LEFT OUTER JOIN Inspect.series series
                    ON study.uid = series.studyUID
                LEFT OUTER JOIN Inspect.image image
                    ON series.uid = image.seriesUID
        FOR XML AUTO, ROOT(N'studyList')
    )
    DECLARE @fileList XML =
    (
        SELECT * FROM v_candidateFile
        FOR XML RAW(N'file'), ROOT(N'fileList')
    )

    IF(@studyList IS NOT NULL)  SET @xml.modify('insert sql:variable("@studyList") into (/inspect)[1]')
    IF(@fileList IS NOT NULL)   SET @xml.modify('insert sql:variable("@fileList") into (/inspect)[1]')

    SELECT @xml;
GO
```

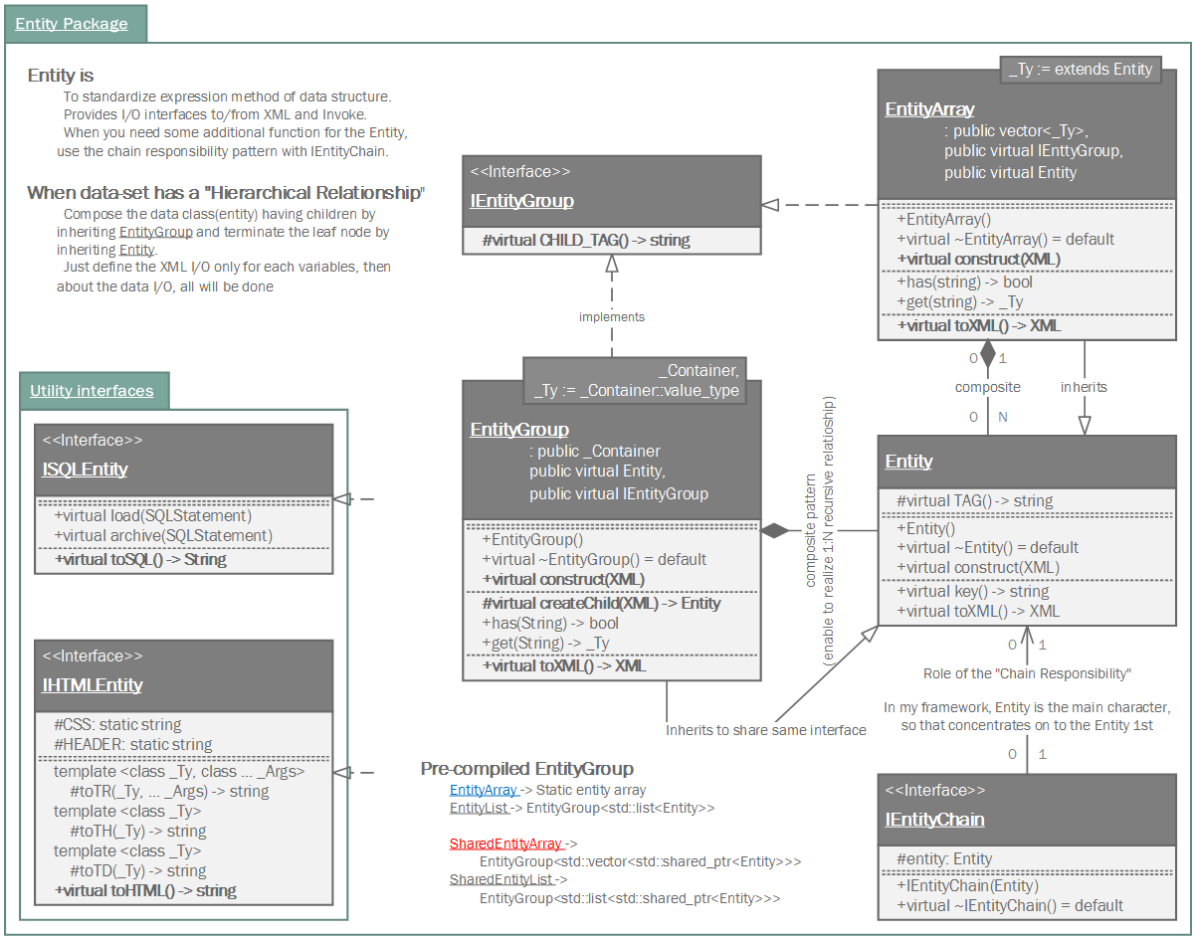
Warning

EntityGroup contains children entity elements as type of pointer. Because children entity objects are not serialized and referenced by pointer, its iteration and accessment is not fast. If it needs higher performance, then use EntityArray (static array for children entity) instead.

[Inherited]

Entity is a class for standardization of expression method using on network I/O by XML. If Invoke is a standard message protocol of Samchon Framework which must be kept, Entity is a recommended semi-protocol of message for expressing a data class. Following the semi-protocol Entity is not imposed but encouraged.

As we could get advantages from standardization of message for network I/O with Invoke, we can get additional advantage from standardizing expression method of data class with Entity. We do not need to know a part of network communication. Thus, with the Entity, we can only concentrate on entity's own logics and relationships between another entities. Entity does not need to how network communications are being done.



Example source

```
1 #include <iostream>
2
3 #include <samchon/protocol/Entity.hpp>
4 #include <samchon/protocol/IHTMLEntity.hpp>
5 #include <samchon/protocol/SharedEntityArray.hpp>
6
7 #include <samchon/library/XML.hpp>
8
9 #ifdef _WIN64
10 #   ifdef _DEBUG
11 #       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
12 #   else
13 #       pragma comment(lib, "x64/Release/SamchonFramework.lib")
14 #   endif
15 #else
16 #   ifdef _DEBUG
17 #       pragma comment(lib, "Debug/SamchonFramework.lib")
18 #   else
19 #       pragma comment(lib, "Release/SamchonFramework.lib")
20 #   endif
21 #endif
22
23 using namespace std;
24 using namespace samchon::library;
25 using namespace samchon::protocol;
26
27 class Member
28     : public Entity, public virtual IHTMLEntity
29 {
30 protected:
31     typedef Entity super;
32
33     string id;
34     string name;
35     int age;
36     int grade;
37
38 public:
39     /* ----------------------------------------------------------------------
40         CONSTRUCTORS
```

```
    ------------------------------------------------------------------- */
    Member()
        : super(), IHTMLEntity()
    {
    };
    Member(const string &id, const string &name, int age, int grade)
        : super(), IHTMLEntity()
    {
        this->id = id;
        this->name = name;
        this->age = age;
        this->grade = grade;
    };
    virtual ~Member() = default;

    virtual void construct(shared_ptr<XML> xml) override
    {
        this->id = xml->getProperty("id");
        this->name = xml->getProperty("name");
        this->age = xml->getProperty<int>("age");
        this->grade = xml->getProperty<int>("grade");
    };

    /* ------------------------------------------------------------------
        GETTERSs
    ------------------------------------------------------------------- */
    virtual auto key() const -> std::string override
    {
        return this->id;
    };

    /* ------------------------------------------------------------------
        XML EXPORTERS
    ------------------------------------------------------------------- */
    virtual auto TAG() const -> string override
    {
        return "member";
    };
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("id", id);
        xml->setProperty("name", name);
        xml->setProperty("age", age);
        xml->setProperty("grade", grade);

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        return toTR(id, name, age, grade);
    };
};

class MemberArray
    : public SharedEntityArray<Member>,
    public virtual IHTMLEntity
{
protected:
    typedef SharedEntityArray<Member> super;

    string application;
    int department;
    Member *chief;

public:
    /* ------------------------------------------------------------------
        CONSTRUCTORS
    ------------------------------------------------------------------- */
    MemberArray()
        : super(), IHTMLEntity()
    {
        this->chief = nullptr;
    };
    virtual ~MemberArray() = default;

    // You don't need to consider children(Member) objects
    // Just concentrate on constructing MemberArray's own member variables
    virtual void construct(shared_ptr<XML> xml) override
    {
        super::construct(xml);

        this->application = xml->getProperty("application");
        this->department = xml->getProperty<int>("department");

        if(xml->hasProperty("chief") == true && this->has( xml->getProperty("chief") ) == true)
            this->chief = this->get( xml->getProperty("cheif") ).get();
```

```
    };

protected:
    //FACTORY METHOD FOR MEMBER
    virtual auto createChild(shared_ptr<XML> = nullptr) -> Member* override
    {
        return new Member();
    };

    /* ---------------------------------------------------------------------
        XML EXPORTERS
    --------------------------------------------------------------------- */
public:
    virtual auto TAG() const -> string override
    {
        return "memberArray";
    };
    virtual auto CHILD_TAG() const -> string override
    {
        return "member";
    };

    // You don't need to consider children(Member) objects
    // Just concentrate on expressing MemberArray's own member variables
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("application", application);
        xml->setProperty("department", department);

        if(chief != nullptr)
            xml->setProperty("cheif", chief->key());

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        string html = "<table>\n";
        html += toTH("ID", "Name", "Age", "Grade") + "\n";

        for (size_t i = 0; i < 2; i++)
            html += at(i)->toHTML() + "\n";

        html += "</table>";
        return move(html);
    };
};

void main()
{
    string str = string("") +
        "<memberArray application='framework' department='7' cheif='samchon'>\n" +
        "   <member id='samchon' name='Jeongho Nam' age='27' grade='5' />" +
        "   <member id='submaster' name='No Name' age='100' grade='4' />" +
        "   <member id='john' name='John Doe' age='33' grade='2' />" +
        "   <member id='bad_man' name='Bad Man' age='44' grade='-1' />" +
        "   <member id='guest' name='Guest' age='0' grade='0' />" +
        "</memberArray>";
    shared_ptr<XML> xml(new XML(str));

    MemberArray memberArray;
    memberArray.construct(xml);

    memberArray.emplace_back(new Member("freshman", "A fresh man", 20, 2));
    memberArray.emplace_back(new Member("senior", "A senior", 70, 2));

    cout << memberArray.toXML()->toString() << endl << endl;
    cout << memberArray.toHTML() << endl;
    system("pause");
}
```

Result of the example

```
D:\OneDrive\Project\Samchon\framework\cpp\Debug\entity.exe
<memberArray application="framework" department="7">
        <member age="27" grade="5" id="samchon" name="Jeongho" />
        <member age="100" grade="4" id="submaster" name="No" />
        <member age="33" grade="2" id="john" name="John" />
        <member age="44" grade="-1" id="bad_man" name="Bad" />
        <member age="0" grade="0" id="guest" name="Guest" />
        <member age="20" grade="2" id="freshman" name="A fresh man" />
        <member age="70" grade="2" id="senior" name="A senior" />
</memberArray>

<table>
        <th>
                <td>ID</td>
                <td>Name</td>
                <td>Age</td>
                <td>Grade</td>
        </th>
        <tr>
                <td>samchon</td>
                <td>Jeongho</td>
                <td>27</td>
                <td>5</td>
        </tr>
        <tr>
                <td>submaster</td>
                <td>No</td>
                <td>100</td>
                <td>4</td>
        </tr>
</table>
Press any key to continue . . .
```

Note

I say repeatedly. Expression method of Entity is recommended, but not imposed. It's a semi protocol for network I/O but not a essential protocol must be kept. The expression method of Entity, using on network I/O, is expressed by XML string.

If your own network system has a critical performance issue on communication data class, it would be better to using binary communication (with ByteArray or boost::serialization). Don't worry about the problem! Invoke also provides methods for binary data (ByteArray).

See also

protocol

Author

Jeongho Nam

6.36.2 Member Function Documentation

6.36.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [inline], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

Reimplemented in DistributedSystem, NTCriteria, ExternalSystem, ParallelSystem, Invoke, DistributedSystemArray, DistributedSystemArrayMediator, ParallelSystemArrayMediator, NTParameter, ExternalClientArray, ExternalServer, DistributedClientArrayMediator, ParallelClientArrayMediator, DistributedClientArray, ParallelServer, and DistributedServer.

6.36.2.2 virtual auto createChild (std::shared_ptr< library::XML >) -> entity_type * [protected], [pure virtual]

Factory method of a child Entity.

EntityGroup::createChild() is a factory method creating a new child Entity which is belonged to the EntityGroup. This method is called by EntityGroup::construct(). The children construction methods Entity::construct() will be called by abstract method of the EntityGroup::construct().

Returns

A new child Entity belongs to EntityGroup.

Implemented in NTCriteria, Invoke, NTParameter, NTParameterArray, DSInvokeHistoryList, PRInvokeHistoryArray, and InvokeHistoryArray.

6.36.2.3 auto has (const std::string & *key*) const -> bool [inline]

Indicates whether a container has an object having the specified identifier.

Parameters

key	An identifier of an Entity

Returns

If there's the object then true, otherwise false

6.36.2.4 auto get (const std::string & *key*) -> value_type& [inline]

Access the element by specified identifier(key).

Parameters

key	the identifier of the element wants to access

Returns

The element having the key, or throw exception if there is none.

6.36.2.5 auto get (const std::string & *key*) const -> const value_type& [inline]

Access the const element by specified identifier(key).

Parameters

key	the identifier of the element wants to access

Returns

The const element having the key, or throw exception if there is none.

6.36.2.6 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [inline],[virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

Reimplemented in Invoke, DistributedSystem, NTCriteria, ExternalSystem, DistributedSystemArray, ParallelSystem, NTParameter, DistributedSystemArrayMediator, ParallelSystemArrayMediator, External↩ ClientArray, ExternalServer, FTFolder, DistributedClientArrayMediator, ParallelClientArrayMediator, DistributedClientArray, ParallelServer, and DistributedServer.

The documentation for this class was generated from the following file:

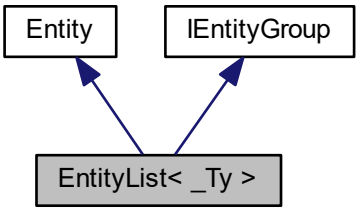
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/EntityGroup.hpp

6.37 EntityList< _Ty > Class Template Reference

An Entity and a static list containing Entity objects.

```
#include <EntityList.hpp>
```

Collaboration diagram for EntityList< _Ty >:



Public Member Functions

- EntityList ()
  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML > xml) override
  Construct data of the Entity from an XML object.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the EntityList.

Additional Inherited Members

6.37.1 Detailed Description

template<typename _Ty>class samchon::protocol::EntityList< _Ty >

An Entity and a static list containing Entity objects.

Template Parameters

_Ty	A type of children Entity. Must be a class derived from an Entity.

EntityList is a static array containing children objects derived from an Entity class.

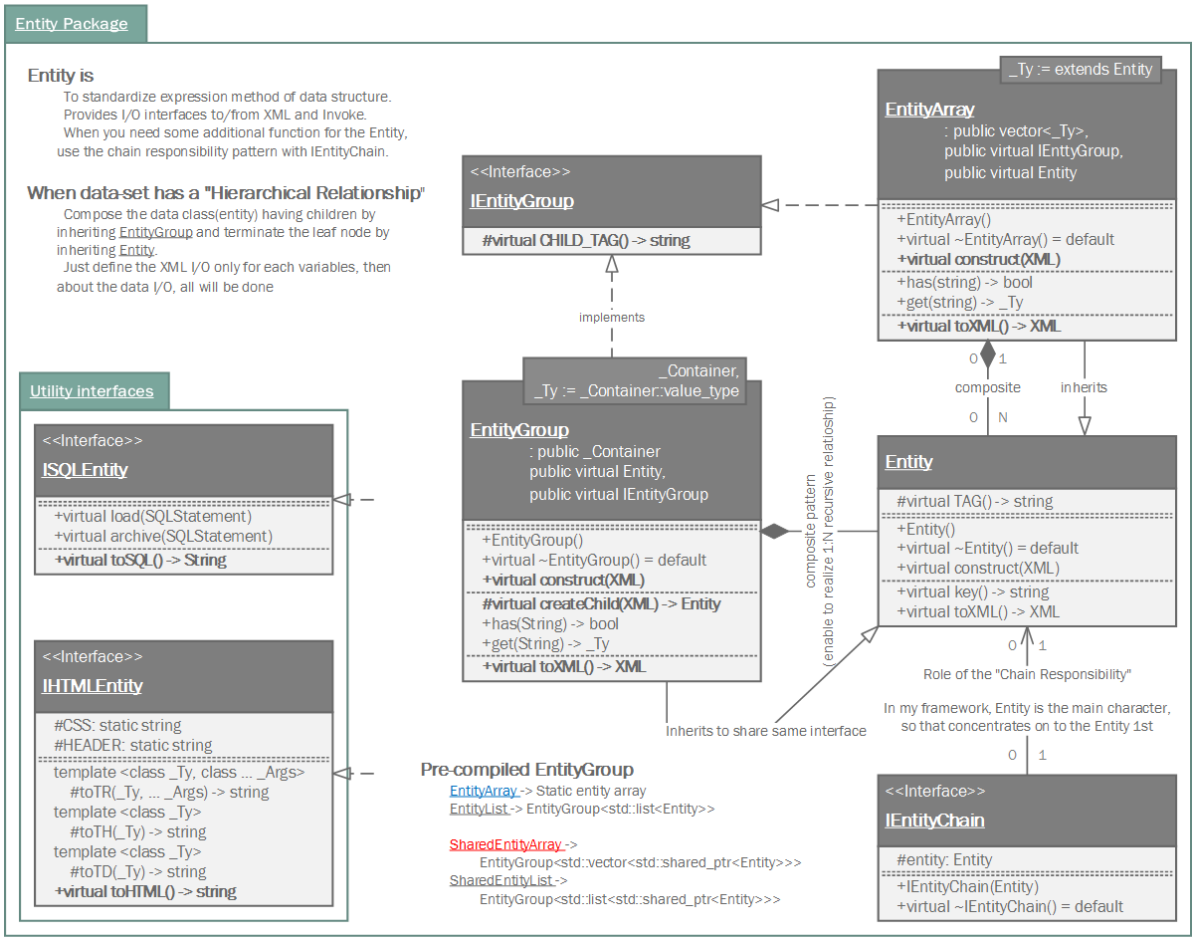
EntityList, its allocated memory space for each child is fixed and cannot contain derived a entity class from the child entity type. It has advantages on performance by direct accessment and static memory allocation for child, however, its increasement of performance is not eminent.

- EntityList<Animal> is specified
- Dog is extended from the Animal
- EntityList<Animal> cannot contain Dog. You can try insertion of the Dog class into the EntityList, but only memory size of Animal is allocated, so that derived member variables and methods are all truncated.

[Inherited]

Entity is a class for standardization of expression method using on network I/O by XML. If Invoke is a standard message protocol of Samchon Framework which must be kept, Entity is a recommended semi-protocol of message for expressing a data class. Following the semi-protocol Entity is not imposed but encouraged.

As we could get advantages from standardization of message for network I/O with Invoke, we can get additional advantage from standardizing expression method of data class with Entity. We do not need to know a part of network communication. Thus, with the Entity, we can only concentrate on entity's own logics and relationships between another entities. Entity does not need to how network communications are being done.



Example source

```
#include <iostream>

#include <samchon/protocol/Entity.hpp>
#include <samchon/protocol/IHTMLEntity.hpp>
#include <samchon/protocol/SharedEntityArray.hpp>

#include <samchon/library/XML.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;

class Member
    : public Entity, public virtual IHTMLEntity
{
protected:
    typedef Entity super;

    string id;
    string name;
    int age;
    int grade;

public:
    /* ----------------------------------------------------------------------
```

```
        CONSTRUCTORS
    ------------------------------------------------------------------- */
    Member()
        : super(), IHTMLEntity()
    {
    };
    Member(const string &id, const string &name, int age, int grade)
        : super(), IHTMLEntity()
    {
        this->id = id;
        this->name = name;
        this->age = age;
        this->grade = grade;
    };
    virtual ~Member() = default;

    virtual void construct(shared_ptr<XML> xml) override
    {
        this->id = xml->getProperty("id");
        this->name = xml->getProperty("name");
        this->age = xml->getProperty<int>("age");
        this->grade = xml->getProperty<int>("grade");
    };

    /* -------------------------------------------------------------------
        GETTERSs
    ------------------------------------------------------------------- */
    virtual auto key() const -> std::string override
    {
        return this->id;
    };

    /* -------------------------------------------------------------------
        XML EXPORTERS
    ------------------------------------------------------------------- */
    virtual auto TAG() const -> string override
    {
        return "member";
    };
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("id", id);
        xml->setProperty("name", name);
        xml->setProperty("age", age);
        xml->setProperty("grade", grade);

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        return toTR(id, name, age, grade);
    };
};

class MemberArray
    : public SharedEntityArray<Member>,
    public virtual IHTMLEntity
{
protected:
    typedef SharedEntityArray<Member> super;

    string application;
    int department;
    Member *chief;

public:
    /* -------------------------------------------------------------------
        CONSTRUCTORS
    ------------------------------------------------------------------- */
    MemberArray()
        : super(), IHTMLEntity()
    {
        this->chief = nullptr;
    };
    virtual ~MemberArray() = default;

    // You don't need to consider children(Member) objects
    // Just concentrate on constructing MemberArray's own member variables
    virtual void construct(shared_ptr<XML> xml) override
    {
        super::construct(xml);

        this->application = xml->getProperty("application");
        this->department = xml->getProperty<int>("department");

        if(xml->hasProperty("chief") == true && this->has( xml->getProperty("chief") ) == true)
```

```
            this->chief = this->get( xml->getProperty("cheif") ).get();
    };

protected:
    //FACTORY METHOD FOR MEMBER
    virtual auto createChild(shared_ptr<XML> = nullptr) -> Member* override
    {
        return new Member();
    };

    /* ------------------------------------------------------------------
        XML EXPORTERS
    ------------------------------------------------------------------ */
public:
    virtual auto TAG() const -> string override
    {
        return "memberArray";
    };
    virtual auto CHILD_TAG() const -> string override
    {
        return "member";
    };

    // You don't need to consider children(Member) objects
    // Just concentrate on expressing MemberArray's own member variables
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("application", application);
        xml->setProperty("department", department);

        if(chief != nullptr)
            xml->setProperty("cheif", chief->key());

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        string html = "<table>\n";
        html += toTH("ID", "Name", "Age", "Grade") + "\n";

        for (size_t i = 0; i < 2; i++)
            html += at(i)->toHTML() + "\n";

        html += "</table>";
        return move(html);
    };
};

void main()
{
    string str = string("") +
        "<memberArray application='framework' department='7' cheif='samchon'>\n" +
        "   <member id='samchon' name='Jeongho Nam' age='27' grade='5' />" +
        "   <member id='submaster' name='No Name' age='100' grade='4' />" +
        "   <member id='john' name='John Doe' age='33' grade='2' />" +
        "   <member id='bad_man' name='Bad Man' age='44' grade='-1' />" +
        "   <member id='guest' name='Guest' age='0' grade='0' />" +
        "</memberArray>";
    shared_ptr<XML> xml(new XML(str));

    MemberArray memberArray;
    memberArray.construct(xml);

    memberArray.emplace_back(new Member("freshman", "A fresh man", 20, 2));
    memberArray.emplace_back(new Member("senior", "A senior", 70, 2));

    cout << memberArray.toXML()->toString() << endl << endl;
    cout << memberArray.toHTML() << endl;
    system("pause");
}
```

Result of the example

```
D:\OneDrive\Project\Samchon\framework\cpp\Debug\entity.exe
<memberArray application="framework" department="7">
        <member age="27" grade="5" id="samchon" name="Jeongho" />
        <member age="100" grade="4" id="submaster" name="No" />
        <member age="33" grade="2" id="john" name="John" />
        <member age="44" grade="-1" id="bad_man" name="Bad" />
        <member age="0" grade="0" id="guest" name="Guest" />
        <member age="20" grade="2" id="freshman" name="A fresh man" />
        <member age="70" grade="2" id="senior" name="A senior" />
</memberArray>

<table>
        <th>
                <td>ID</td>
                <td>Name</td>
                <td>Age</td>
                <td>Grade</td>
        </th>
        <tr>
                <td>samchon</td>
                <td>Jeongho</td>
                <td>27</td>
                <td>5</td>
        </tr>
        <tr>
                <td>submaster</td>
                <td>No</td>
                <td>100</td>
                <td>4</td>
        </tr>
</table>
Press any key to continue . . .
```

Note

I say repeatedly. Expression method of Entity is recommended, but not imposed. It's a semi protocol for network I/O but not a essential protocol must be kept. The expression method of Entity, using on network I/O, is expressed by XML string.

If your own network system has a critical performance issue on communication data class, it would be better to using binary communication (with ByteArray or boost::serialization). Don't worry about the problem! Invoke also provides methods for binary data (ByteArray).

See also

protocol

Author

Jeongho Nam

Warning

If a data structure has recursive and hierarchical relationship, never use the EntityList. EntityList's memory allocation is static. The recursive relationship causes infinite memory allocation. It must generates stack overflow.

Use EntityGroup instead.

- class NTCriteria : public EntityList<NTCriteria> causes a stack overflow.
- class NTCriteria : public SharedEntityList<NTCriteria> doesn't cause the stack overflow.

Warning

Unlike EntityArray containing children serialized, EntityList's static memory allocation and accessment for a child, its advantage doesn't seemed to have eminent advantage on performance. As there's not meaningful advantage but has only disadvantage that cannot containing derived objects, I'm planning to depreciate the EntityList in next generation.

Use SharedEntityList instead will be better I think.

6.37.2 Member Function Documentation

6.37.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [inline], [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityList's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityList::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityList::toXML().

Constructs only data of EntityList's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

6.37.2.2 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [inline], [override], [virtual]

Get an XML object represents the EntityList.

Archives the EntityList's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityList::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityList::toXML().

Archives only data of EntityList's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

The documentation for this class was generated from the following file:

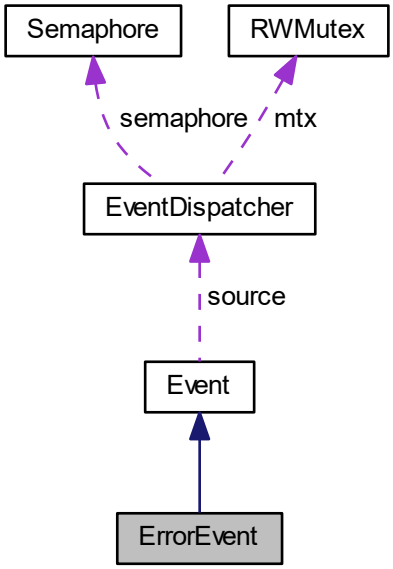
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/EntityList.hpp

6.38 ErrorEvent Class Reference

Event representing an error.

`#include <ErrorEvent.hpp>`

Collaboration diagram for ErrorEvent:



Public Member Functions

- ErrorEvent (EventDispatcher *, const std::string &)

  Construct from source and error-id.
- auto getMessage () const -> std::string

  Get error-id.

Protected Attributes

– std::string message

Error message.

6.38.1 Detailed Description

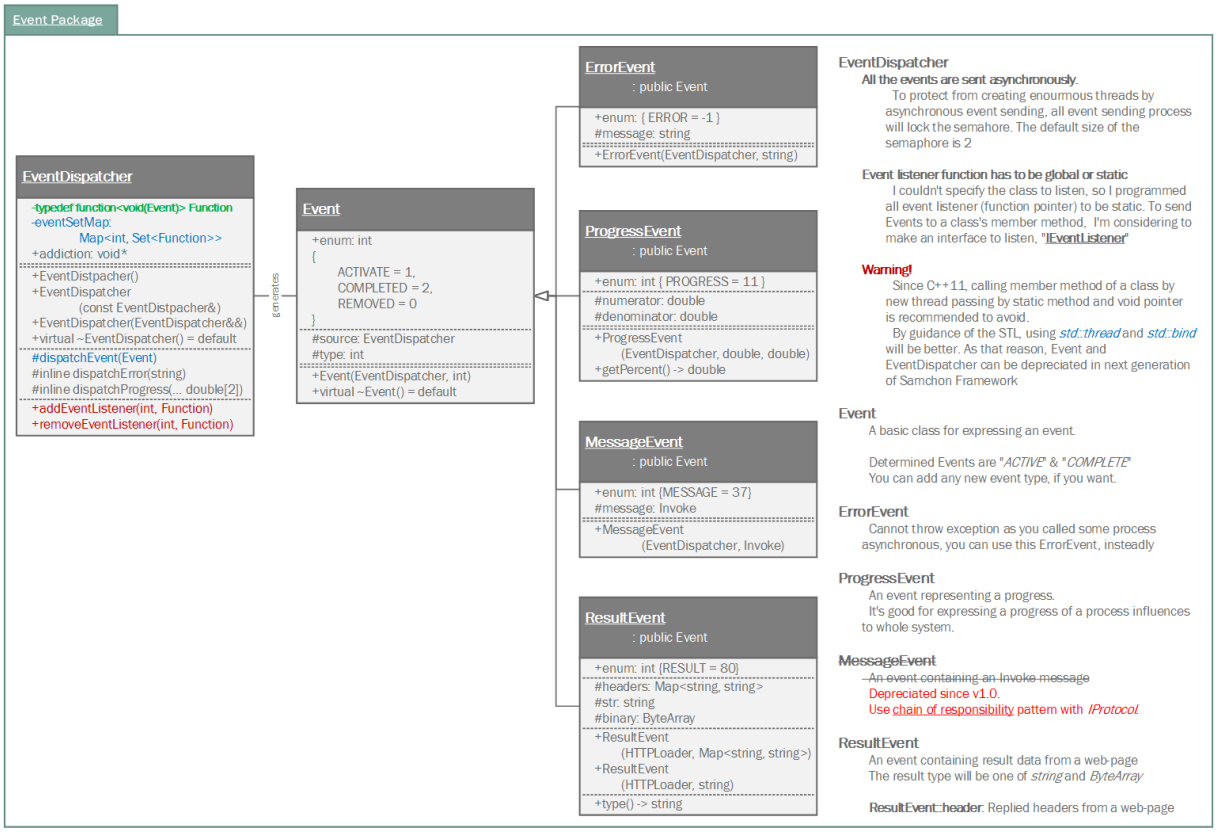
Event representing an error.

[Inherited]

The Event class is used as the base class for the creation of Event objects, which are passed as parameters to event listeners when an event occurs.

The properties of the Event class carry basic information about an event, such as the event's type or source (who made the event) of the event.

For many events, such as the events represented by the Event class constants, this basic information is sufficient. Other events, however, may require more detailed information.



Example source

```
#include <iostream>
#include <thread>
#include <chrono>

#include <samchon/library/EventDispatcher.hpp>
#include <samchon/library/Event.hpp>
#include <samchon/library/ProgressEvent.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
```

```
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace std;
using namespace samchon::library;

class Process
    : public EventDispatcher
{
public:
    Process()
    {

    };

    void run()
    {
        dispatchEvent(shared_ptr<Event>(new Event(this, Event::ACTIVATE)));

        for (size_t i = 0; i < 100; i++)
        {
            this_thread::sleep_for(chrono::seconds(1));
            dispatchEvent(shared_ptr<Event>(new ProgressEvent(this, i + 1, 100)));
        }

        dispatchEvent(shared_ptr<Event>(new Event(this, Event::COMPLETE)));
    };
};

void handleActivate(shared_ptr<Event> event)
{
    cout << "Activated" << endl;
}
void handleComplete(shared_ptr<Event> event)
{
    cout << "Completed" << endl;
}
void handleProgress(shared_ptr<Event> event)
{
    cout << dynamic_pointer_cast<ProgressEvent>(event)->getPercent() * 100.0 << "%" << endl;
}

void main()
{
    Process process;

    process.addEventListener(Event::ACTIVATE, handleActivate);
    process.addEventListener(Event::COMPLETE, handleComplete);
    process.addEventListener(ProgressEvent::PROGRESS, handleProgress);

    process.run();
    system("pause");
}
```

Note

Event is a candidate to be depreciated.

Since C++11, calling member method of a class by a new thread passing by static method and using void pointer are recommeded to avoid. As the reason, using *std::thread* and *std::bind will* be better.

- std::thread: http://www.cplusplus.com/reference/thread/thread/
- std::bind: http://www.cplusplus.com/reference/functional/bind/

See also

samchon::library

Author

Jeongho Nam

6.38.2 Constructor & Destructor Documentation

6.38.2.1 ErrorEvent (EventDispatcher *, const std::string &)

Construct from source and error-id.

The event object owns its source and type

Parameters

source	Source of the event; who made the event
id	An error-id

The documentation for this class was generated from the following files:

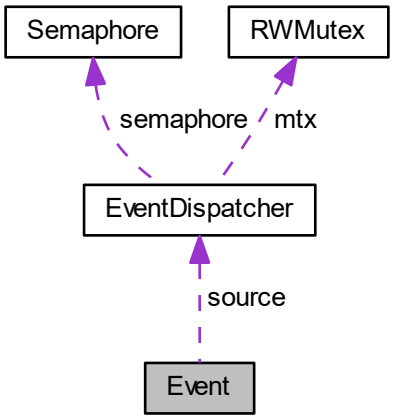
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/ErrorEvent.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/ErrorEvent.cpp

6.39 Event Class Reference

Represent an event running on background.

`#include <Event.hpp>`

Collaboration diagram for Event:



Public Member Functions

- Event (EventDispatcher *, int)

  Construct from source and type.
- auto getSource () const -> EventDispatcher *

  Get source of the Event.
- auto getType () const -> int

  Get type of the Event.

Protected Attributes

- EventDispatcher * source

  Source of the event.
- int type

  Type of the event.

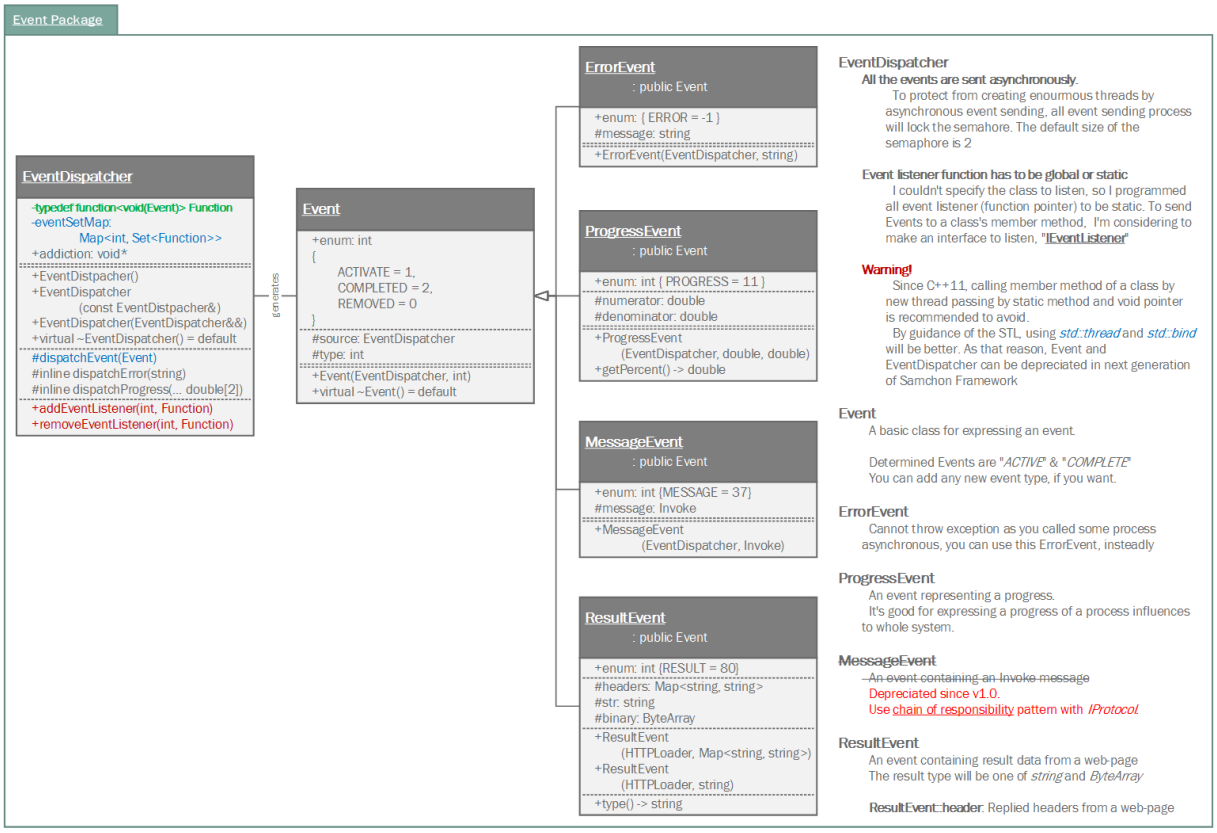
6.39.1 Detailed Description

Represent an event running on background.

The Event class is used as the base class for the creation of Event objects, which are passed as parameters to event listeners when an event occurs.

The properties of the Event class carry basic information about an event, such as the event's type or source (who made the event) of the event.

For many events, such as the events represented by the Event class constants, this basic information is sufficient. Other events, however, may require more detailed information.



Example source

```
1 #include <iostream>
2 #include <thread>
3 #include <chrono>
4
5 #include <samchon/library/EventDispatcher.hpp>
6 #include <samchon/library/Event.hpp>
7 #include <samchon/library/ProgressEvent.hpp>
8
9 #ifdef _WIN64
10 #   ifdef _DEBUG
11 #       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
12 #   else
13 #       pragma comment(lib, "x64/Release/SamchonFramework.lib")
14 #   endif
15 #else
16 #   ifdef _DEBUG
```

```
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace std;
using namespace samchon::library;

class Process
    : public EventDispatcher
{
public:
    Process()
    {

    };

    void run()
    {
        dispatchEvent(shared_ptr<Event>(new Event(this, Event::ACTIVATE)));

        for (size_t i = 0; i < 100; i++)
        {
            this_thread::sleep_for(chrono::seconds(1));
            dispatchEvent(shared_ptr<Event>(new ProgressEvent(this, i + 1, 100)));
        }

        dispatchEvent(shared_ptr<Event>(new Event(this, Event::COMPLETE)));
    };
};

void handleActivate(shared_ptr<Event> event)
{
    cout << "Activated" << endl;
}
void handleComplete(shared_ptr<Event> event)
{
    cout << "Completed" << endl;
}
void handleProgress(shared_ptr<Event> event)
{
    cout << dynamic_pointer_cast<ProgressEvent>(event)->getPercent() * 100.0 << "%" << endl;
}

void main()
{
    Process process;

    process.addEventListener(Event::ACTIVATE, handleActivate);
    process.addEventListener(Event::COMPLETE, handleComplete);
    process.addEventListener(ProgressEvent::PROGRESS, handleProgress);

    process.run();
    system("pause");
}
```

Note

Event is a candidate to be depreciated.

Since C++11, calling member method of a class by a new thread passing by static method and using void pointer are recommeded to avoid. As the reason, using *std::thread* and *std::bind will* be better.

- std::thread: http://www.cplusplus.com/reference/thread/thread/
- std::bind: http://www.cplusplus.com/reference/functional/bind/

See also

samchon::library

Author

Jeongho Nam

6.39.2 Constructor & Destructor Documentation

6.39.2.1 Event (EventDispatcher * *source*, int *type*)

Construct from source and type.

The event object owns its source and type

Parameters

source	Source of the event; who made the event
type	Type of the event

The documentation for this class was generated from the following files:

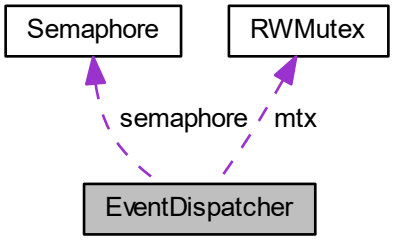
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Event.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Event.cpp

6.40 EventDispatcher Class Reference

Abstract class for dispatching Event.

`#include <EventDispatcher.hpp>`

Collaboration diagram for EventDispatcher:



Public Member Functions

- EventDispatcher ()
  Default Constructor.
- EventDispatcher (const EventDispatcher &eventDispatcher)
  Copy Constructor.
- EventDispatcher (EventDispatcher &&eventDispatcher)
  Move Constructor.
- void addEventListener (int, void(*listener)(std::shared_ptr< Event >))
  Register an event listener.
- void removeEventListener (int, void(*listener)(std::shared_ptr< Event >))
  Remove a registered event listener.

Protected Member Functions

- auto dispatchEvent (std::shared_ptr< Event >) -> bool
  Dispatches an event to all listeners.
- auto dispatchProgressEvent (size_t x, size_t size) -> bool
  Convenient method of dispatching a progress event.

Private Attributes

- std::map< int, std::set< void(*)(std::shared_ptr< Event >)> > eventSetMap

  A container storing listeners.
- RWMutex mtx

  A rw_mutex for concurrency.
- Semaphore semaphore

  A semaphore for restricting thread size.

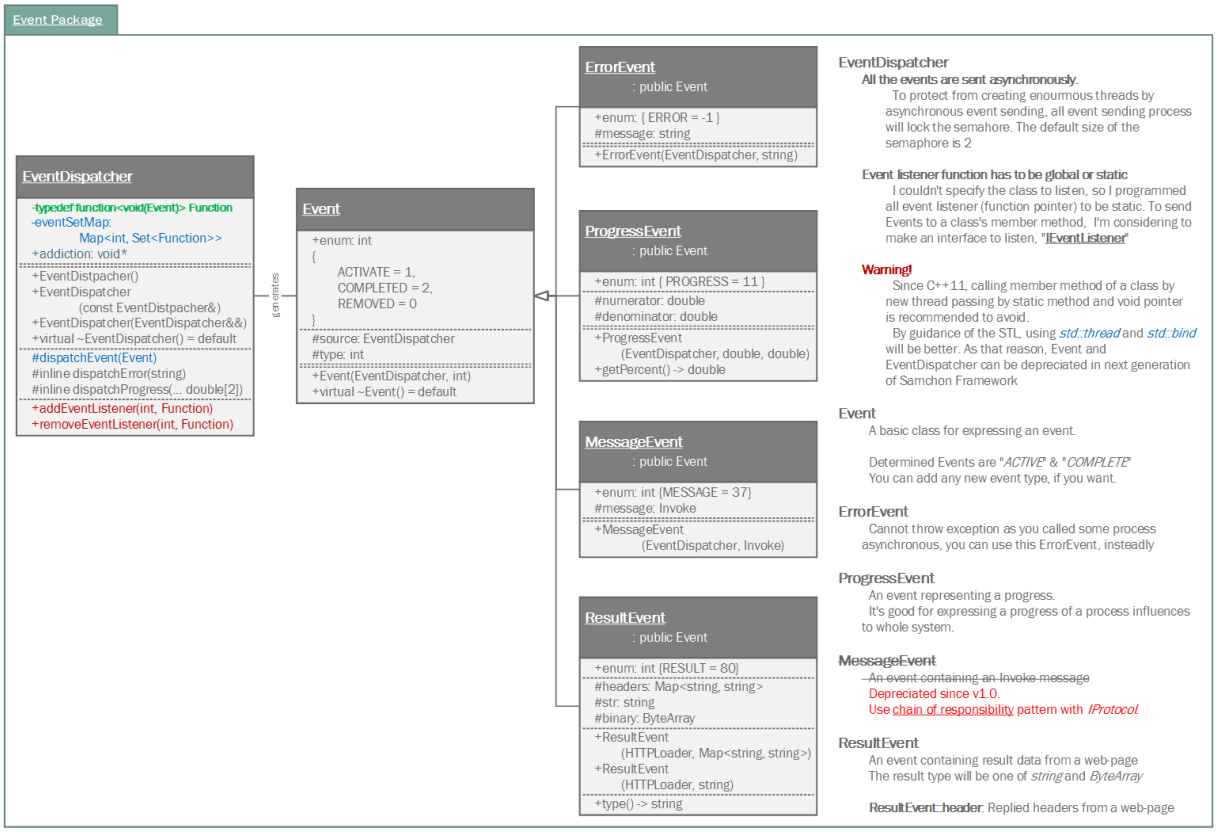
6.40.1 Detailed Description

Abstract class for dispatching Event.

EventDispatcher is the base class for all classes that dispatch events.

All the events are sent asynchronously. To avoid from creating tooo enourmouse threads dispatching events, all event sending processes will acuiqre a semaphore. The default permitted size of the semaphore is 2.

- Number of thread pools used to sending events is 2.



Example source

```
#include <iostream>
#include <thread>
#include <chrono>

#include <samchon/library/EventDispatcher.hpp>
#include <samchon/library/Event.hpp>
#include <samchon/library/ProgressEvent.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
```

```
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace std;
using namespace samchon::library;

class Process
    : public EventDispatcher
{
public:
    Process()
    {

    };

    void run()
    {
        dispatchEvent(shared_ptr<Event>(new Event(this, Event::ACTIVATE)));

        for (size_t i = 0; i < 100; i++)
        {
            this_thread::sleep_for(chrono::seconds(1));
            dispatchEvent(shared_ptr<Event>(new ProgressEvent(this, i + 1, 100)))
    ;
        }

        dispatchEvent(shared_ptr<Event>(new Event(this, Event::COMPLETE)));
    };
};

void handleActivate(shared_ptr<Event> event)
{
    cout << "Activated" << endl;
}
void handleComplete(shared_ptr<Event> event)
{
    cout << "Completed" << endl;
}
void handleProgress(shared_ptr<Event> event)
{
    cout << dynamic_pointer_cast<ProgressEvent>(event)->getPercent() * 100.0 << "%" << endl;
}

void main()
{
    Process process;

    process.addEventListener(Event::ACTIVATE, handleActivate);
    process.addEventListener(Event::COMPLETE, handleComplete);
    process.addEventListener(ProgressEvent::PROGRESS, handleProgress);

    process.run();
    system("pause");
}
```

Note

EventDispatcher is a candidate to be depreciated.

Since C++11, calling member method of a class by a new thread passing by static method and using void pointer are recommeded to avoid. As the reason, using *std::thread* and *std::bind* will be better.

- std::thread: `http://www.cplusplus.com/reference/thread/thread/`
- std::bind: `http://www.cplusplus.com/reference/functional/bind/`

See also

samchon::library

Author

Jeongho Nam

6.40.2 Constructor & Destructor Documentation

6.40.2.1 EventDispatcher (const EventDispatcher & *eventDispatcher*)

Copy Constructor.

Copying an EventDispatcher instance does not copy the event listeners attached to it. (If your newly created node needs an event listener, you must attach the listener after creating the node.)

Parameters

eventDispatcher	The object to copy

6.40.2.2 EventDispatcher (EventDispatcher && *eventDispatcher*)

Move Constructor.

Parameters

eventDispatcher	The object to move

6.40.3 Member Function Documentation

6.40.3.1 void addEventListener (int *type*, void(∗)(std::shared_ptr< Event >) *listener*)

Register an event listener.

Registers an event listener object with an EventDispatcher object so that the listener receives notification of an event.

Warning

> Copying an EventDispatcher instance does not copy the event listeners attached to it. (If your newly created node needs an event listener, you must attach the listener after creating the node.) However, if you move an EventDispatcher instance, the event listeners attached to it move along with it.
> If you no longer need an event listener, remove it by calling removeEventListener, or EventDispatcher already try to send events to the no longer needed listener and it can cause some confliction.

Parameters

type	The type of event.
listener	The listener function processes the event.

6.40.3.2 void removeEventListener (int *type*, void(∗)(std::shared_ptr< Event >) *listener*)

Remove a registered event listener.

Removes a listener from the EventDispatcher object. If there is no matching listener registered with the EventDispatcher object, a call to this method has no effect

Parameters

type	The type of event.
listener	The listener function to remove.

6.40.3.3 auto dispatchEvent (std::shared_ptr< Event >) -> bool [protected]

Dispatches an event to all listeners.

Dispatches an event into the event flow in the background. The Event::source is the EventDispatcher object upon which the dispatchEvent.

Parameters

event	The Event object that is dispatched into the event flow.

Returns

Whether there's some listener to listen the event

6.40.3.4 auto dispatchProgressEvent (size_t *x*, size_t *size*) -> bool [protected]

Convenient method of dispatching a progress event.

Dispatches a progress event into the event flow in the background The Event::source is the EventDispatcher object upon with the dispatchProgressEvent

Parameters

x	The number of current progress
size	The number of total progress

Returns

Whether there's some listener to listen the progress event

See also

ProgressEvent
EventDispatcher::dispatchEvent

The documentation for this class was generated from the following files:

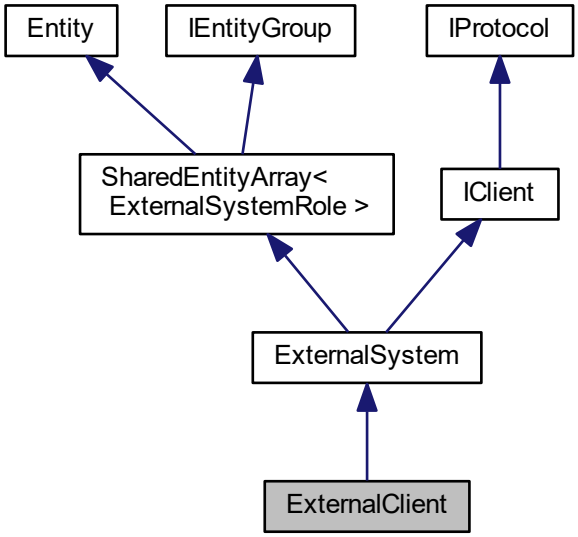
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/EventDispatcher.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/EventDispatcher.cpp

6.41 ExternalClient Class Reference

A network driver for an external client.

```
#include <ExternalClient.hpp>
```

Collaboration diagram for ExternalClient:



Public Member Functions

– ExternalClient ()

Default Constructor.

Additional Inherited Members

6.41.1 Detailed Description

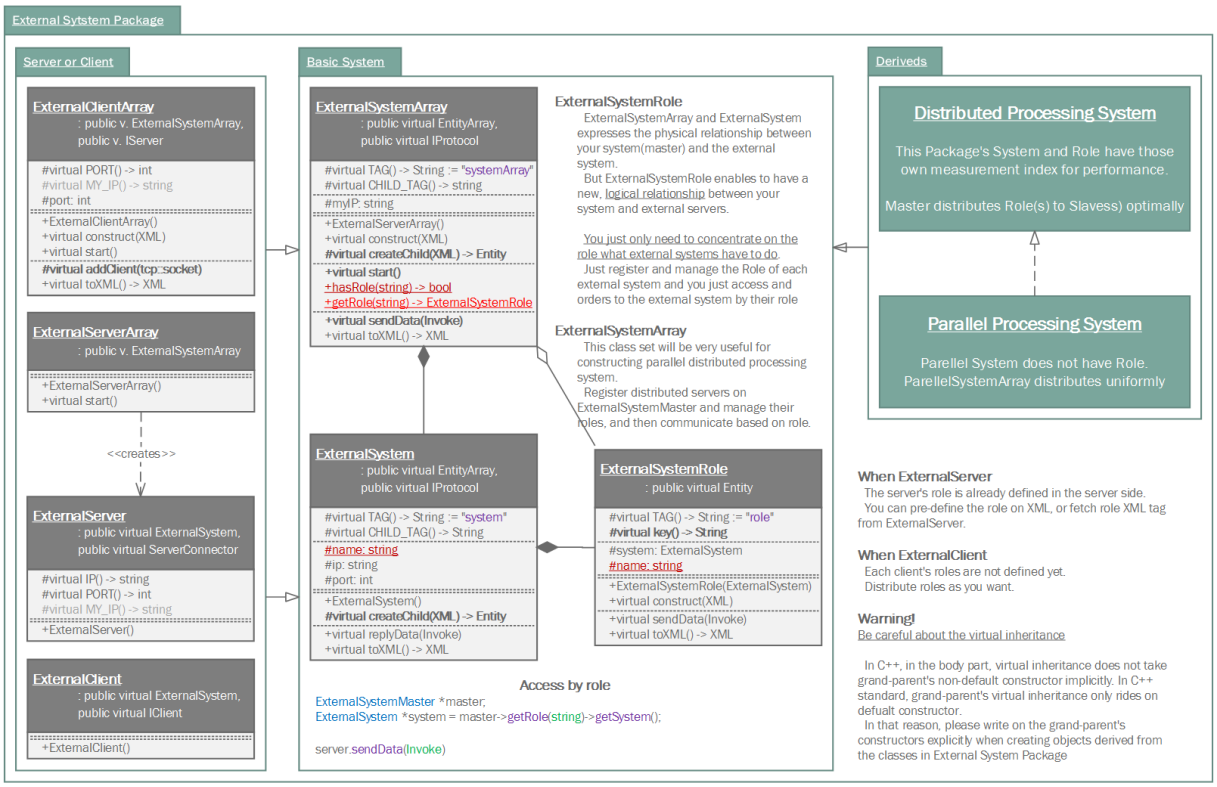
A network driver for an external client.

ExternalClient is an ExternalSystem specialized in client driver.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystem↩ Role.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩ SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an External↩ System. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following files:

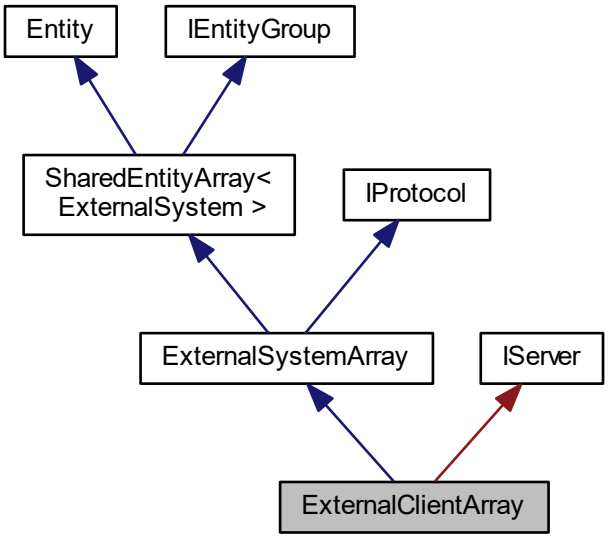
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalClient.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalClient.cpp

6.42 ExternalClientArray Class Reference

An array of ExternalClient(s)

```
#include <ExternalClientArray.hpp>
```

Collaboration diagram for ExternalClientArray:



Public Member Functions

- ExternalClientArray ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual void start () override

  Start interaction.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Protected Member Functions

- virtual auto PORT () const -> int

  Port number of the server.
- virtual auto MY_IP () const -> std::string

  (optional) Server's IP
- virtual void addClient (Socket *) override

  Handling connection of a physical client.

Protected Attributes

- std::string myIP

  A custom ip address of my system to bind.
- int port

  A port number for accepting external clients.

Additional Inherited Members

6.42.1 Detailed Description

An array of ExternalClient(s)

ExternalClientArray is an ExternalSystemArray specialized in clients' driver.

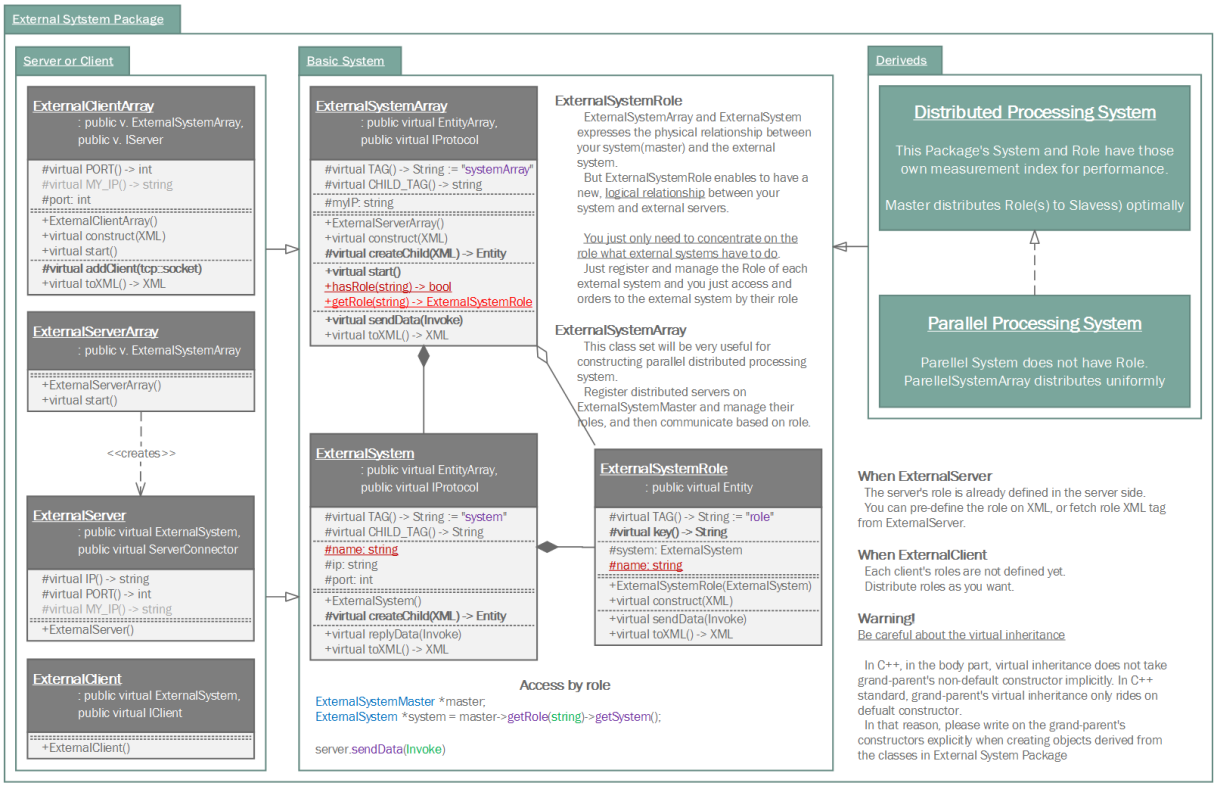
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.42.2 Member Function Documentation

6.42.2.1 void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in DistributedClientArrayMediator, ParallelClientArrayMediator, and DistributedClientArray.

6.42.2.2 void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Implements ExternalSystemArray.

Reimplemented in DistributedClientArrayMediator, ParallelClientArrayMediator, and DistributedClientArray.

6.42.2.3 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in DistributedClientArrayMediator, ParallelClientArrayMediator, and DistributedClientArray.

The documentation for this class was generated from the following files:

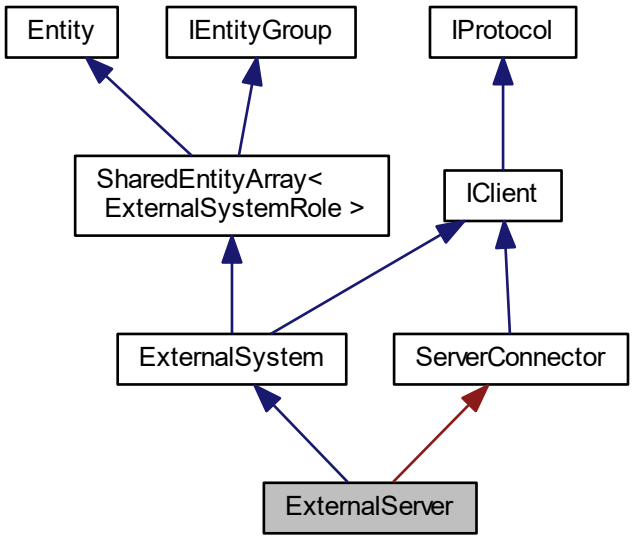
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalClientArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalClientArray.cpp

6.43 ExternalServer Class Reference

A network driver for an external server.

`#include <ExternalServer.hpp>`

Collaboration diagram for ExternalServer:



Public Member Functions

- ExternalServer ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.

- virtual void start () override
  Start interaction.
- virtual auto getIP () const -> std::string override
  Destinatio IP.
- virtual auto getPort () const -> int override
  Destination port.
- virtual auto getMyIP () const -> std::string override
  (optional) My IP, if you want to bind
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the EntityGroup.

Protected Attributes

- std::string myIP
  A custom ip address of my system to bind.

Additional Inherited Members

6.43.1 Detailed Description

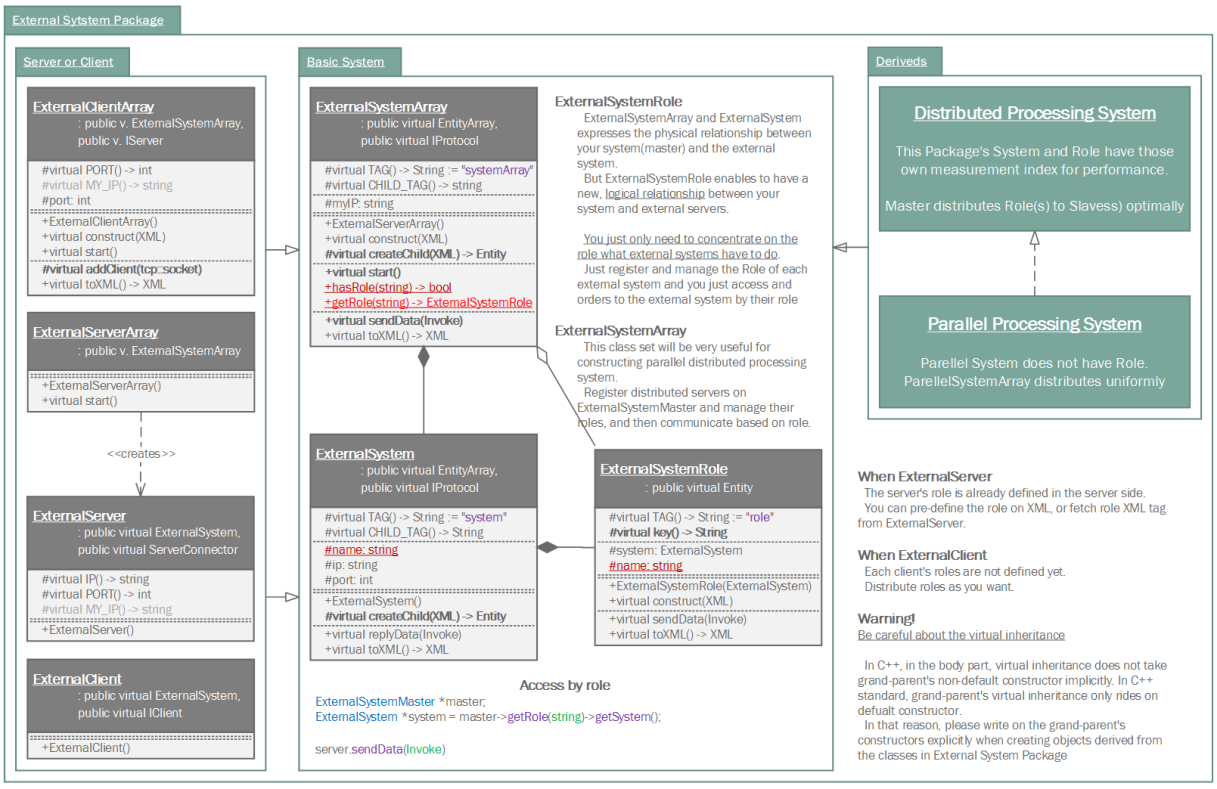
A network driver for an external server.

ExternalServer is an ExternalSystem specialized in server driver.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

6.43.2 Member Function Documentation

6.43.2.1 void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from ExternalSystem.

Reimplemented in ParallelServer, and DistributedServer.

6.43.2.2 void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with an external system.

If an external systems are a server, starts connection and listening Inovoke message, else clients, just starts listening only. You also can addict your own procudures of starting the driver, but if you directly override method of abstract ExternalSystem, be careful about virtual inheritance.

Implements ExternalSystem.

6.43.2.3 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ExternalSystem.

Reimplemented in ParallelServer, and DistributedServer.

The documentation for this class was generated from the following files:

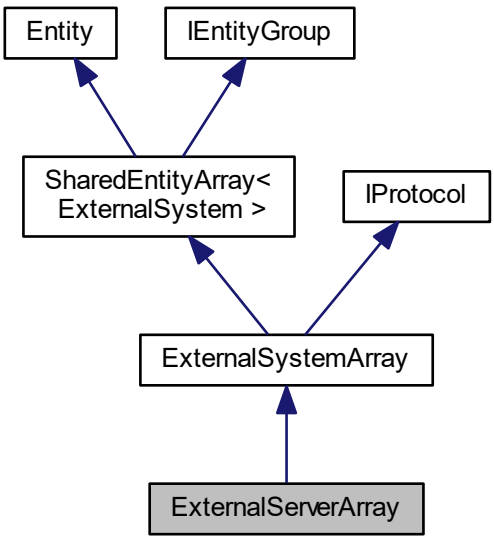
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalServer.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalServer.cpp

6.44 ExternalServerArray Class Reference

An array of ExternalServer(s)

```
#include <ExternalServerArray.hpp>
```

Collaboration diagram for ExternalServerArray:



Public Member Functions

- virtual void start () override
  Start interaction.

Additional Inherited Members

6.44.1 Detailed Description

An array of ExternalServer(s)

ExternalServerArray is an ExternalSystemArray specialized in servers' driver.

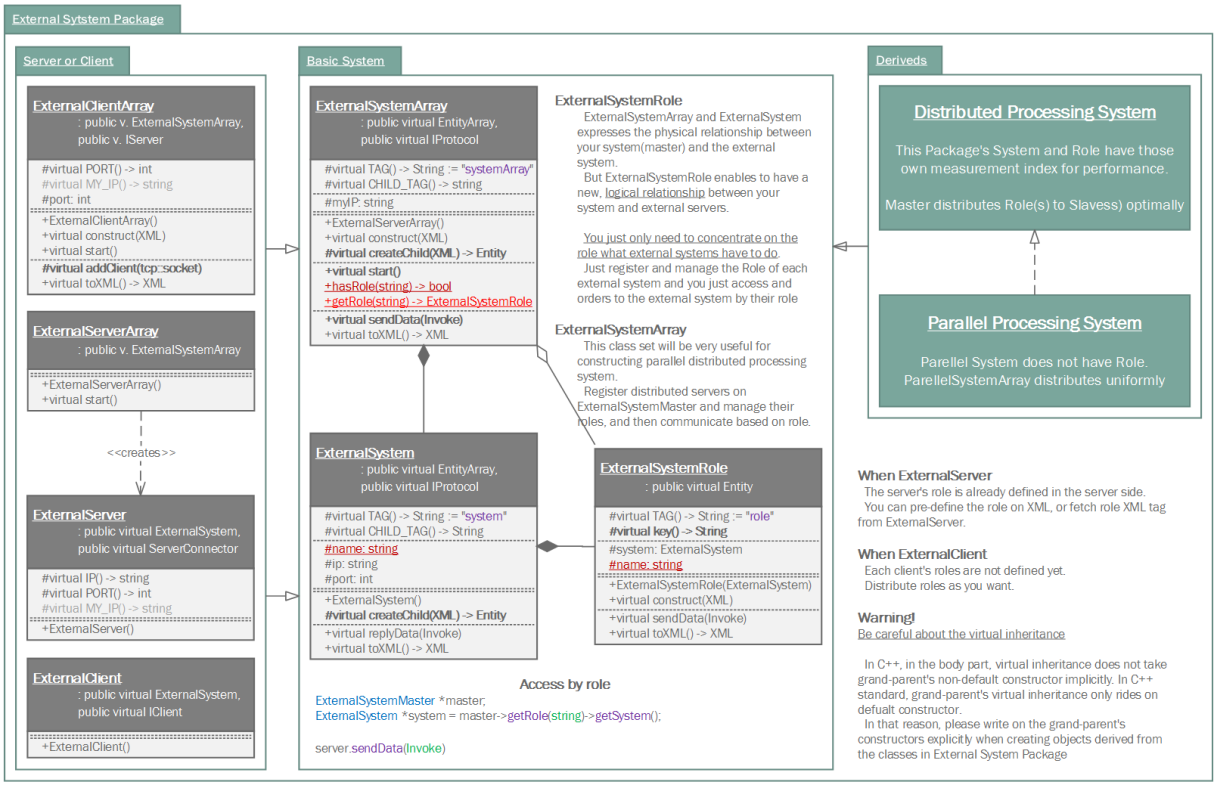
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.44.2 Member Function Documentation

6.44.2.1 void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Implements ExternalSystemArray.

Reimplemented in DistributedServerArrayMediator, ParallelServerArrayMediator, and DistributedServer↩ Array.

The documentation for this class was generated from the following files:

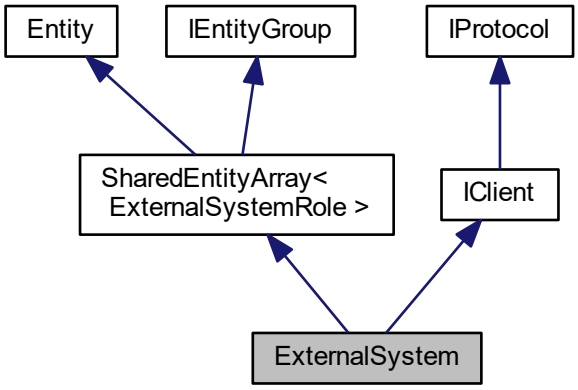
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalServerArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalServerArray.cpp

6.45 ExternalSystem Class Reference

A network driver for an external system.

`#include <ExternalSystem.hpp>`

Collaboration diagram for ExternalSystem:



Public Member Functions

- ExternalSystem ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual void start ()=0

  Start interaction.
- virtual auto key () const -> std::string override

  Get a key that can identify the Entity uniquely.
- virtual void replyData (std::shared_ptr< Invoke >) override

  Handling replied message from an external system.
- virtual auto TAG () const -> std::string override

  A tag name when represented by XML.
- virtual auto CHILD_TAG () const -> std::string override

  A tag name of children.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Protected Attributes

- std::string name

  A name can identify an external system.
- std::string ip

  An ip address of an external system.
- int port

  A port number of an external system.

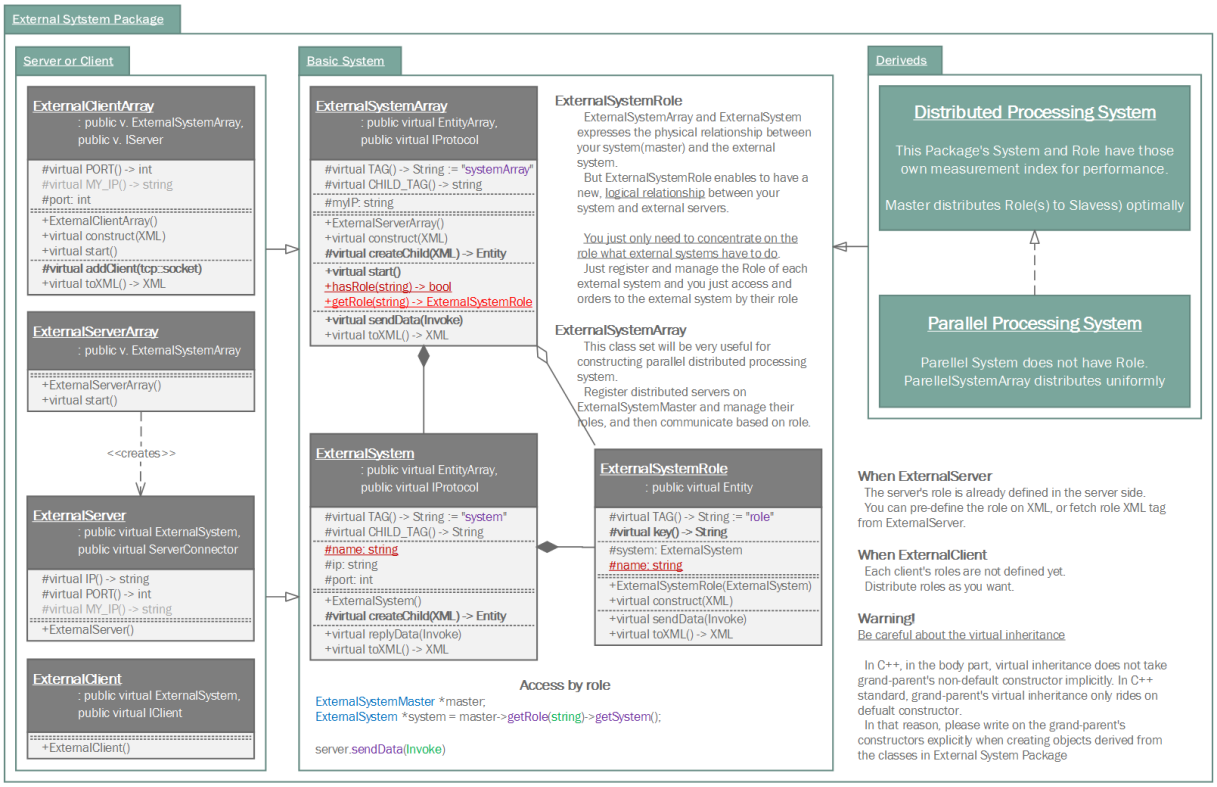
Additional Inherited Members

6.45.1 Detailed Description

A network driver for an external system.

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystem↩ Role.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩ SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an External↩ System. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

6.45.2 Member Function Documentation

6.45.2.1 void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in DistributedSystem, ParallelSystem, ExternalServer, ParallelServer, and DistributedServer.

6.45.2.2 virtual void start ( ) [pure virtual]

Start interaction.

An abstract method starting interaction with an external system.

If an external systems are a server, starts connection and listening Inovoke message, else clients, just starts listening only. You also can addict your own procudures of starting the driver, but if you directly override method of abstract ExternalSystem, be careful about virtual inheritance.

Implemented in ExternalServer.

6.45.2.3 auto key ( ) const -> std::string [override], [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented from Entity.

6.45.2.4 void replyData (std::shared_ptr< Invoke >) [override], [virtual]

Handling replied message from an external system.

If there's a related ExternalSystemRole, shifts responsibility to the ExternalSystemRole.

Parameters

invoke	Replied Invoke message

Reimplemented from IProtocol.

Reimplemented in DistributedSystem, DistributedSlaveSystemMediator, and ParallelSlaveSystemMediator.

6.45.2.5 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

Reimplemented in DistributedSlaveSystemMediator, and ParallelSlaveSystemMediator.

6.45.2.6 auto CHILD_TAG ( ) const -> std::string [override], [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

6.45.2.7 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in DistributedSystem, ParallelSystem, ExternalServer, ParallelServer, and DistributedServer.

6.45.3 Member Data Documentation

6.45.3.1 std::string name [protected]

A name can identify an external system.

The name must be unique in ExternalSystemArray.

The documentation for this class was generated from the following files:

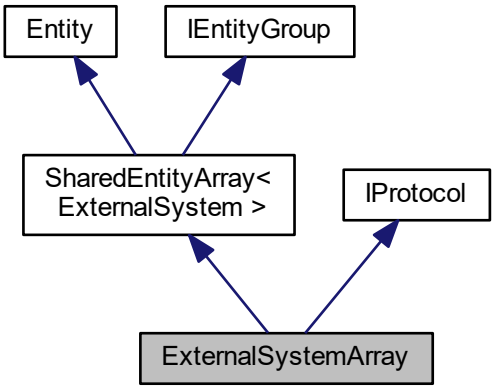
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalSystem.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalSystem.cpp

6.46 ExternalSystemArray Class Reference

An array of ExternalSystem(s).

`#include <ExternalSystemArray.hpp>`

Collaboration diagram for ExternalSystemArray:



Public Member Functions

- ExternalSystemArray ()
  Default Constructor.
- virtual void start ()=0
  Start interaction.
- auto hasRole (const std::string &) const -> bool
  Test whether has a role.
- auto getRole (const std::string &) const -> std::shared_ptr< ExternalSystemRole >
  Get a role.
- virtual void sendData (std::shared_ptr< Invoke >) override
  Sends an Invoke message to related ExternalSystem(s).
- virtual void replyData (std::shared_ptr< Invoke >) override
  Handle replied Invoke message.
- virtual auto TAG () const -> std::string override
  A tag name when represented by XML.
- virtual auto CHILD_TAG () const -> std::string override
  A tag name of children.

Additional Inherited Members

6.46.1 Detailed Description

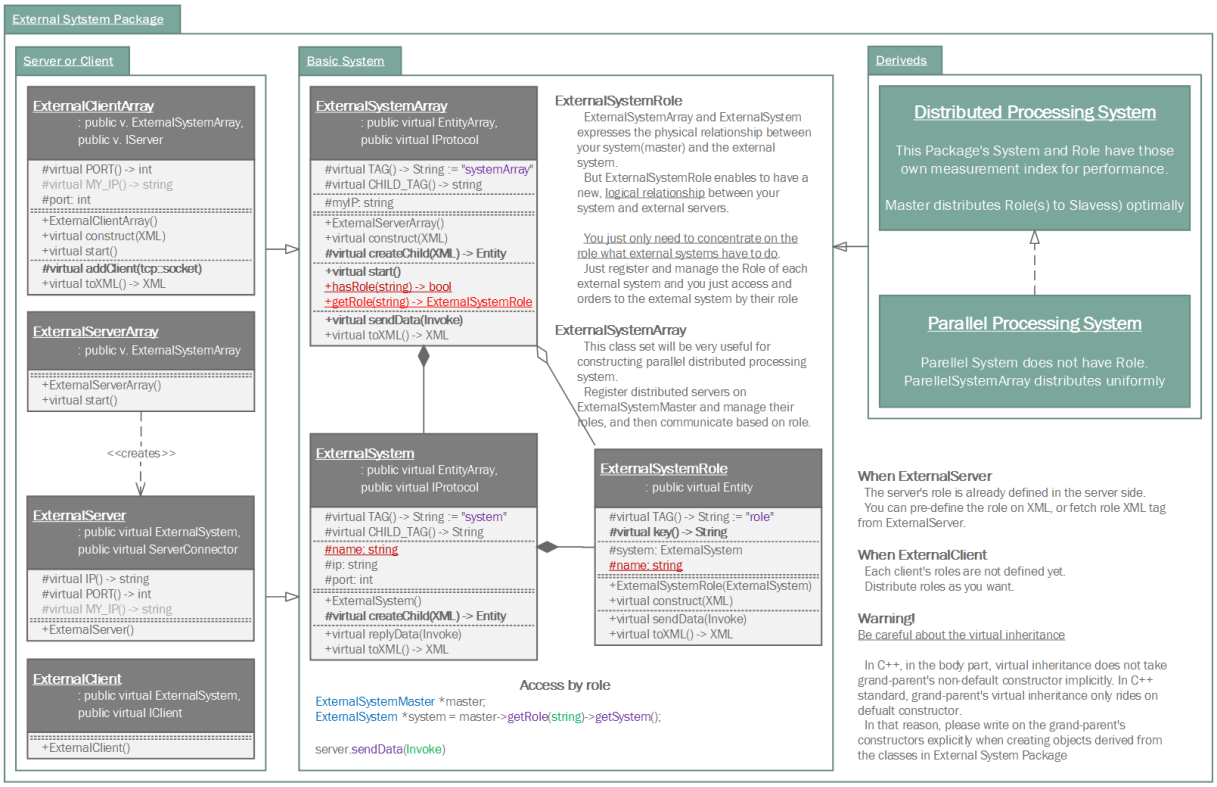
An array of ExternalSystem(s).

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.46.2 Member Function Documentation

6.46.2.1 virtual void start ( ) [pure virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Implemented in DistributedSystemArray, DistributedSystemArrayMediator, ParallelSystemArrayMediator, ExternalClientArray, DistributedClientArrayMediator, ParallelClientArrayMediator, DistributedClientArray, DistributedServerArrayMediator, ParallelServerArrayMediator, DistributedServerArray, and ExternalServer↩Array.

6.46.2.2 auto hasRole (const std::string & *name*) const -> bool

Test whether has a role.

Parameters

name	Name of an ExternalSystemRole

Returns

Whether has or not

6.46.2.3 auto getRole (const std::string & *name*) const -> std::shared_ptr<ExternalSystemRole>

Get a role.

Parameters

name	Name of an ExternalSystemRole

Returns

A shared pointer of specialized role

6.46.2.4 void sendData (std::shared_ptr< Invoke >) [override], [virtual]

Sends an Invoke message to related ExternalSystem(s).

If listener of the Invoke message is registered on some ExternalSystemRole(s), sends the Invoke message to related ExternalSystem(s) (containing the ExternalSystemRole(s)). If cannot find any ExternalSystemRole, sends the Invoke message to all ExternalSystem(s) belongs to the ExternalSystemArray.

Parameters

invoke	Invoke message to send

Reimplemented from IProtocol.

6.46.2.5 void replyData (std::shared_ptr< Invoke >) [override], [virtual]

Handle replied Invoke message.

Nothing is defined in the ExternalSystemArray::replyData(). It's only for avoid the method to have a status "not overriden, abstract".

Parameters

invoke	Invoke message to send

Reimplemented from IProtocol.

6.46.2.6 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.46.2.7 auto CHILD_TAG ( ) const -> std::string [override], [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

The documentation for this class was generated from the following files:

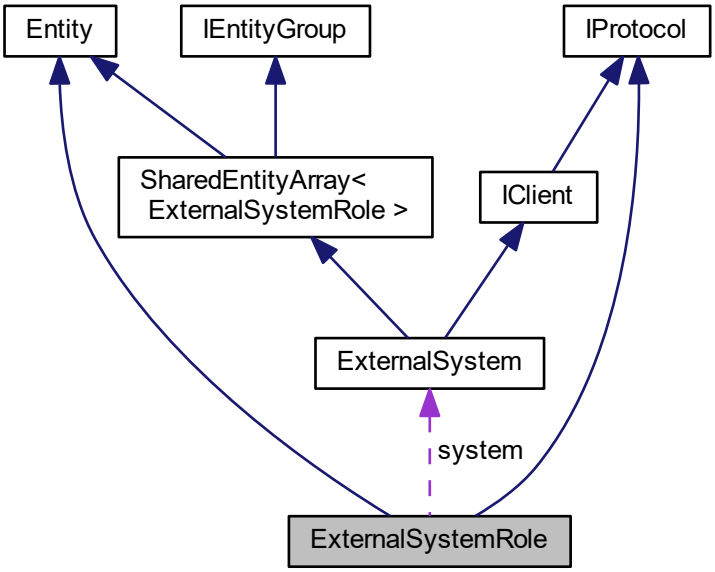
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalSystemArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalSystemArray.cpp

6.47 ExternalSystemRole Class Reference

A role belongs to an external system.

```
#include <ExternalSystemRole.hpp>
```

Collaboration diagram for ExternalSystemRole:



Public Member Functions

- ExternalSystemRole (ExternalSystem *)
  Construct from external system driver.
- virtual void construct (std::shared_ptr< library::XML >) override
  Construct data of the Entity from an XML object.
- auto getSystem () const -> ExternalSystem *
  Get an external system driver.
- virtual auto key () const -> std::string override
  Get a key that can identify the Entity uniquely.
- auto hasSendListener (const std::string &) const -> bool
  Test whether has a listener for send in the role.
- auto hasReplyListener (const std::string &) const -> bool
  Test whether has a listener for reply in the role.
- virtual void sendData (std::shared_ptr< Invoke >) override
  Send a message.
- virtual auto TAG () const -> std::string override
  A tag name when represented by XML.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the Entity.

Protected Attributes

- std::string name
  A name representing the role.
- std::set< std::string > replyListeners
  Listeners to reply in the role.
- std::set< std::string > sendListeners
  Listeners to send in the role.

Private Attributes

– ExternalSystem * system

A driver of external system containing the ExternalSystemRole.

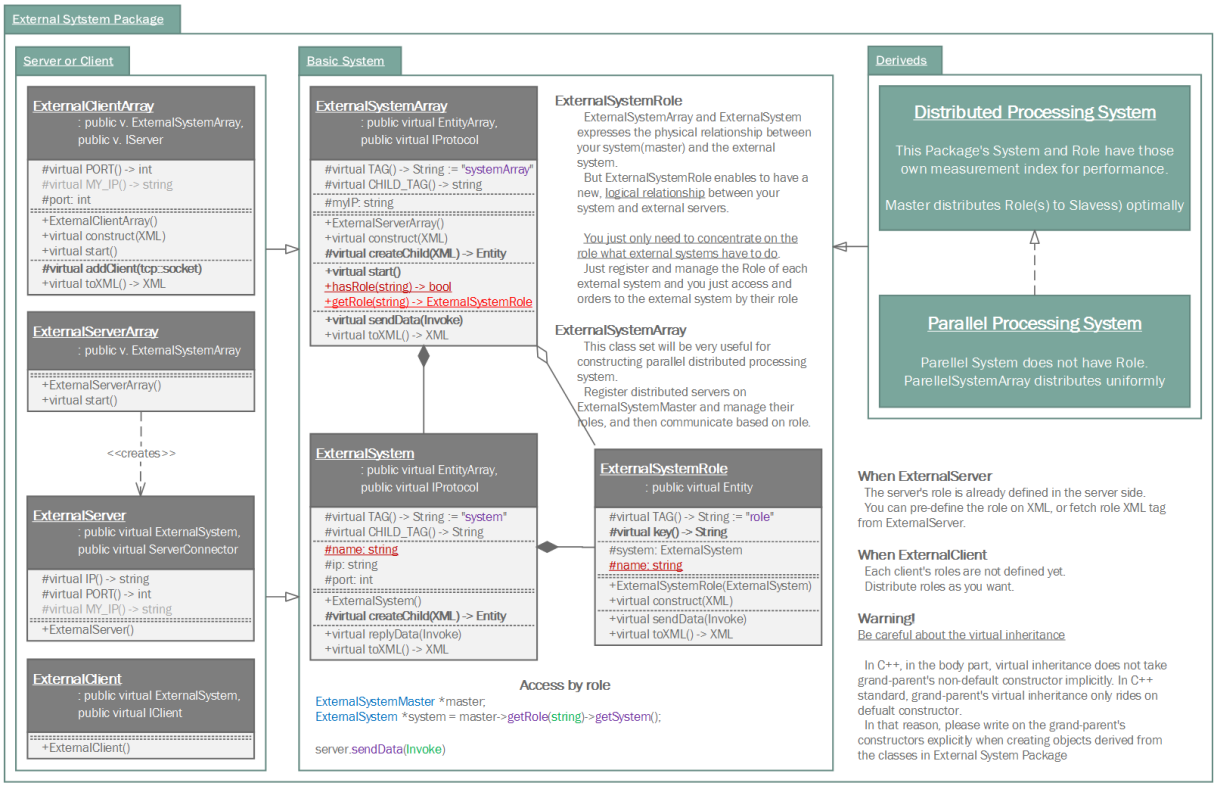
6.47.1 Detailed Description

A role belongs to an external system.

ExternalSystemRole is a 'control' class groupping methods, handling Invoke messages interacting with an external system that the ExternalSystemRole is belonged to, by a subject or unit of a module.

ExternalSystemRole can be a "logical proxy" for an ExternalSystem which is containing the ExternalSystem↩ Role. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystem↩ Role acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.



See also

samchon::protocol

Author

Jeongho Nam

6.47.2 Constructor & Destructor Documentation

6.47.2.1 ExternalSystemRole (ExternalSystem * *system*)

Construct from external system driver.

Parameters

system	A driver of external system the ExternalSystemRole is belonged to.

6.47.3 Member Function Documentation

6.47.3.1 void construct (std::shared_ptr< library::XML >) [override], [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

6.47.3.2 auto key ( ) const -> std::string [override], [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented from Entity.

6.47.3.3 void sendData (std::shared_ptr< Invoke >) [override], [virtual]

Send a message.

Sends Invoke message to a network system or shifts the responsibility to related chain.

Parameters

invoke	An Invoke message to send to a network system

Reimplemented from IProtocol.

Reimplemented in DistributedSystemRole.

6.47.3.4 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.47.3.5 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

Reimplemented in DistributedSystemRole.

The documentation for this class was generated from the following files:

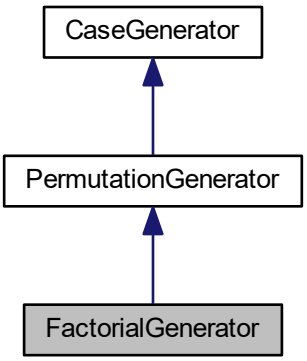
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalSystemRole.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ExternalSystemRole.cpp

6.48 FactorialGenerator Class Reference

Factorical case generator.

`#include <FactorialGenerator.hpp>`

Collaboration diagram for FactorialGenerator:



Public Member Functions

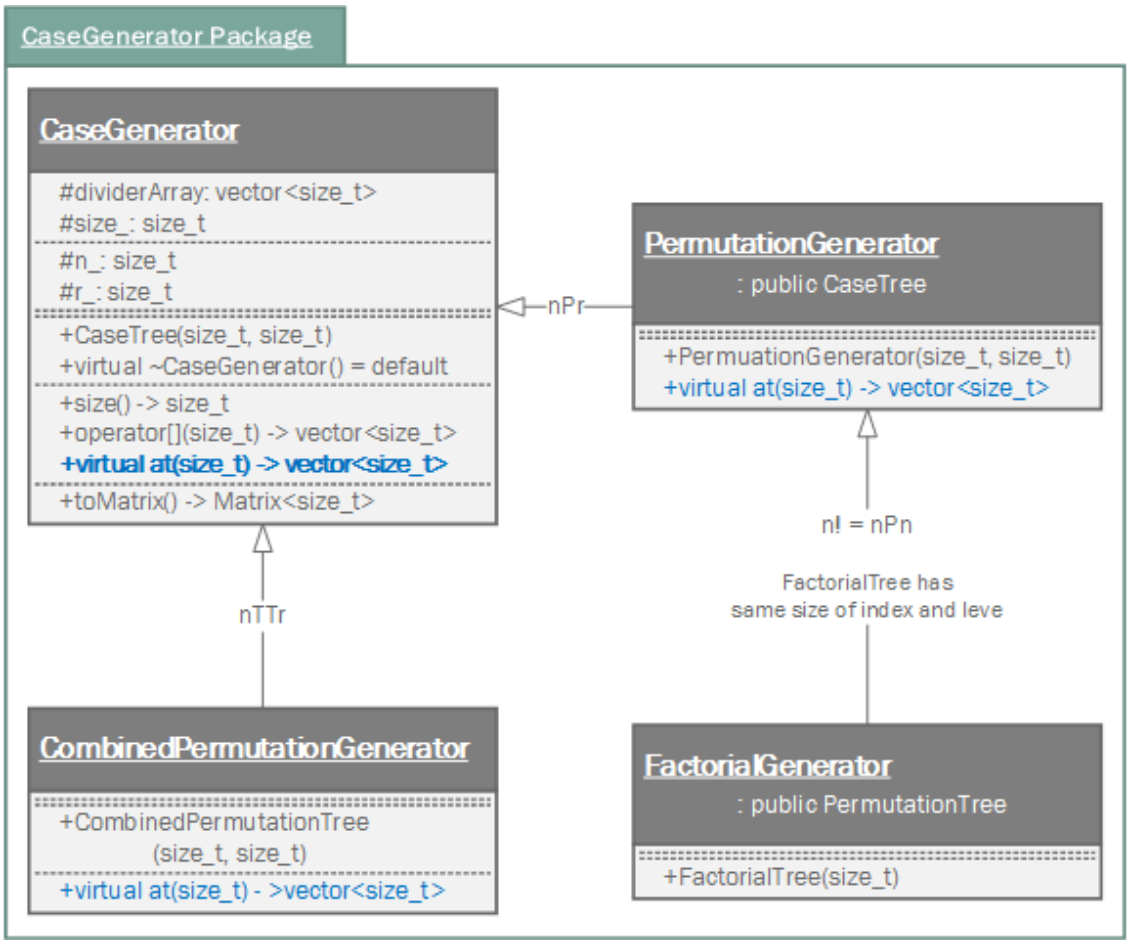
- FactorialGenerator (size_t size)
  Construct from size N.

Additional Inherited Members

6.48.1 Detailed Description

Factorical case generator.

n! = nPn



See also

samchon::library

Author

Jeongho Nam

6.48.2 Constructor & Destructor Documentation

6.48.2.1 FactorialGenerator (size_t *size*)

Construct from size N.

Parameters

n	Size of elements of each case to be arranged.

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FactorialGenerator.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FactorialGenerator.cpp

6.49 FlashPolicyServer Class Reference

A flash policy server.

```
#include <FlashPolicyServer.hpp>
```

Public Member Functions

- FlashPolicyServer ()
  Default constructor.
- FlashPolicyServer (std::shared_ptr< library::XML >)
  Construct from custom policy.
- void openServer ()
  Open server.

Protected Attributes

- std::shared_ptr< library::XML > policy
  cross-domain-policy

Private Member Functions

- void accept (Socket *)
  Accept client.

6.49.1 Detailed Description

A flash policy server.

FlashPolicyServer is a policy server replying cross-domain-policy to flash clients.

Author

Jeongho Nam

6.49.2 Constructor & Destructor Documentation

6.49.2.1 FlashPolicyServer ( )

Default constructor.

The cross-domain-policy is to accept all, any client.

6.49.2.2 FlashPolicyServer (std::shared_ptr< library::XML >)

Construct from custom policy.

Parameters

policy	Custom cross-domain-policy to adjust

6.49.3 Member Function Documentation

6.49.3.1 void accept (Socket *) [private]

Accept client.

Accepts flash client and replies cross-domain-policy

The documentation for this class was generated from the following files:

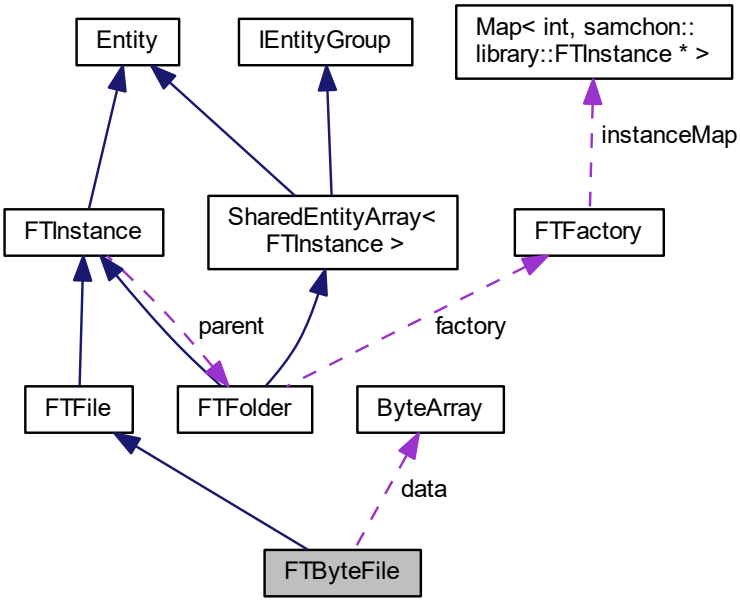
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/FlashPolicyServer.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/FlashPolicyServer.cpp

6.50 FTByteFile Class Reference

A binary file.

`#include <FTByteFile.hpp>`

Collaboration diagram for FTByteFile:



Public Member Functions

- FTByteFile (FTFolder *)

  Construct from parent folder.
- auto getData () -> ByteArray

  Get binaray data of the file.

Protected Attributes

- ByteArray data

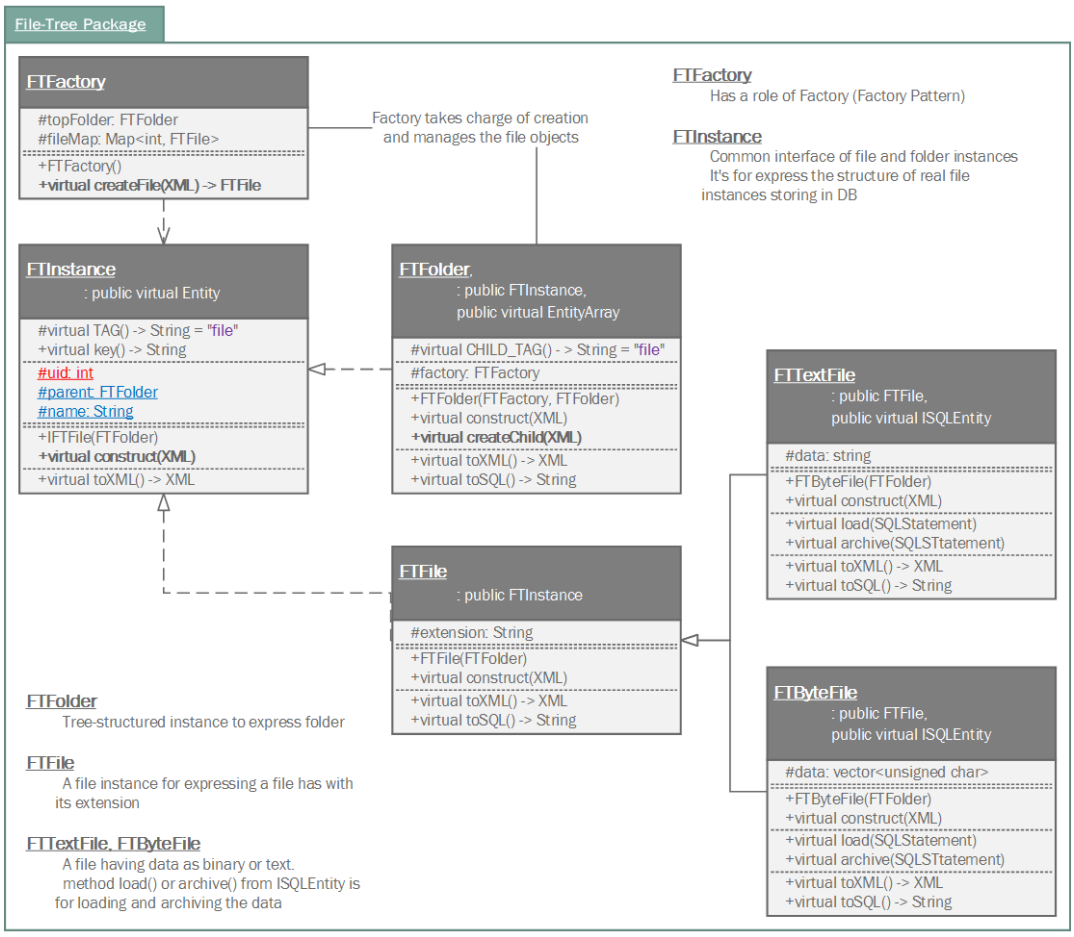
  Binary data of the file.

Additional Inherited Members

6.50.1 Detailed Description

A binary file.

A file having data as binary.



See also

samchon::library

Author

Jeongho Nam

6.50.2 Constructor & Destructor Documentation

6.50.2.1 FTByteFile (FTFolder *)

Construct from parent folder.

Parameters

folder	Belonged folder

The documentation for this class was generated from the following file:

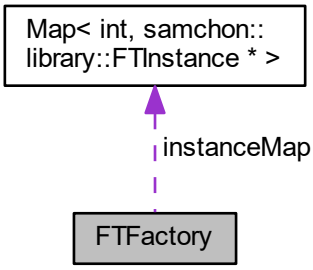
– D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTByteFile.hpp

6.51 FTFactory Class Reference

A Factory and manager for file-tree.

```
#include <FTFactory.hpp>
```

Collaboration diagram for FTFactory:



Public Member Functions

- FTFactory ()

  Default Constructor.
- virtual auto createFile (FTFolder ∗, std::shared_ptr< XML >) -> FTFile ∗=0

  Factory method of a file.
- void registerInstance (FTInstance ∗)

  Register file instance to map.

Protected Attributes

- int application

  An application id.
- int category

  A categori id.
- std::string member

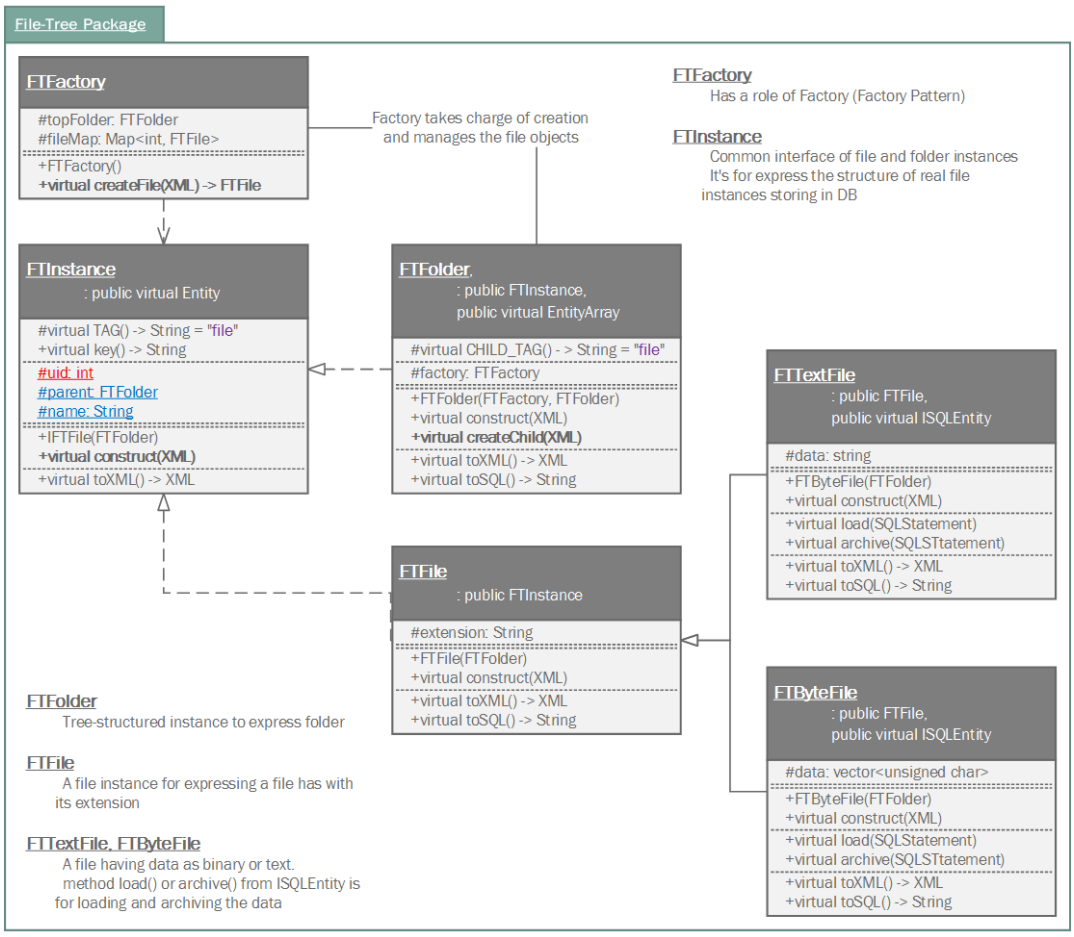
  Account id of a member.
- Map< int, FTInstance ∗ > instanceMap

  Map of files.

6.51.1 Detailed Description

A Factory and manager for file-tree.

FTFactory is a class taking a role of creating FTFile instances and managing them by having their direct pointers. FTFactory also has a role of archiving and loading from DB.



See also

samchon::library

Author

Jeongho Nam

6.51.2 Member Data Documentation

6.51.2.1 int application [protected]

An application id.

An enumeration code of an application that managed FTInstance(s) are belonged to.

6.51.2.2 int category [protected]

A categori id.

An enumeration code of a category that managed FTInstance(s) are belonged to.

6.51.2.3 Map<int, FTInstance*> instanceMap [protected]

Map of files.

- key: uid
- value: pointer of file

The documentation for this class was generated from the following files:

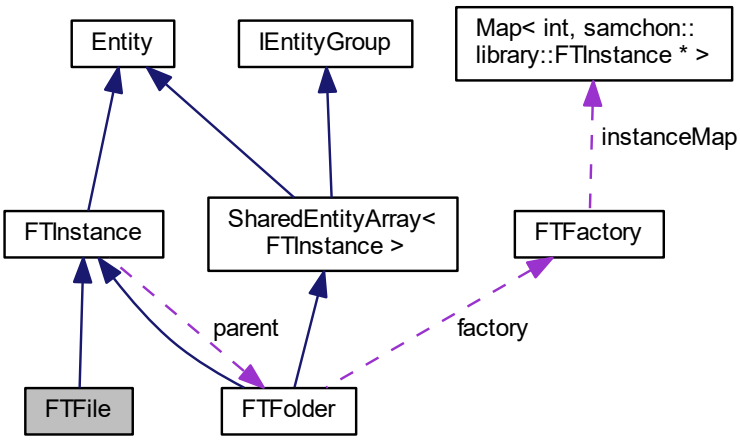
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTFactory.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTFactory.cpp

6.52 FTFile Class Reference

A file.

`#include <FTFile.hpp>`

Collaboration diagram for FTFile:



Public Member Functions

- FTFile (FTFolder *)
  Construct from parent folder.
- auto getExtension () const -> std::string
  Get extension.
- virtual auto toXML () const -> std::shared_ptr< library::XML >
  Get an XML object represents the Entity.

Protected Attributes

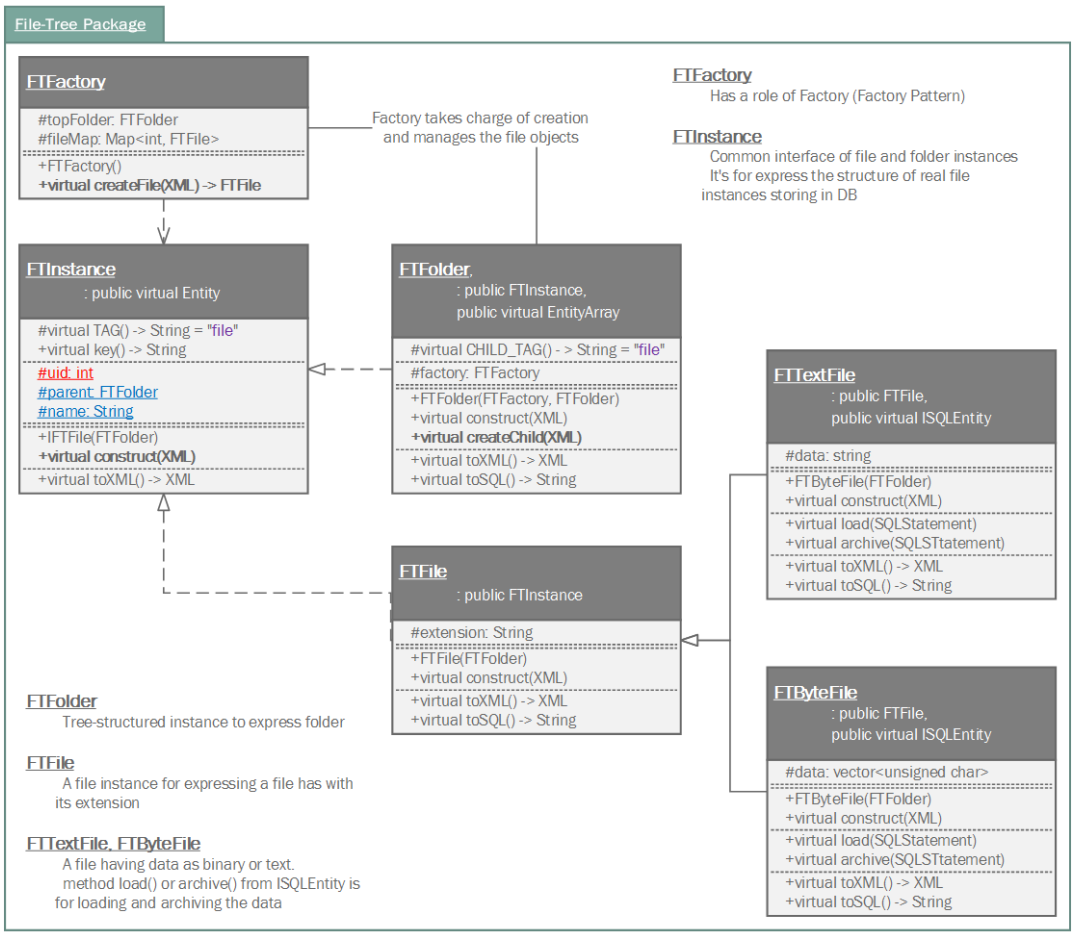
- std::string extension
  An extension of a file.

Additional Inherited Members

6.52.1 Detailed Description

A file.

An abstract class represents a file instance with extension.



See also

samchon::library

Author

Jeongho Nam

6.52.2 Constructor & Destructor Documentation

6.52.2.1 FTFile (FTFolder * *parent*)

Construct from parent folder.

Parameters

folder	Belonged folder

6.52.3 Member Function Documentation

6.52.3.1 auto toXML ( ) const -> std::shared_ptr<library::XML> [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from FTInstance.

Reimplemented in NTFile.

6.52.4 Member Data Documentation

6.52.4.1 std::string extension [protected]

An extension of a file.

FTFile's identifier is composited by name and extension.

The documentation for this class was generated from the following files:

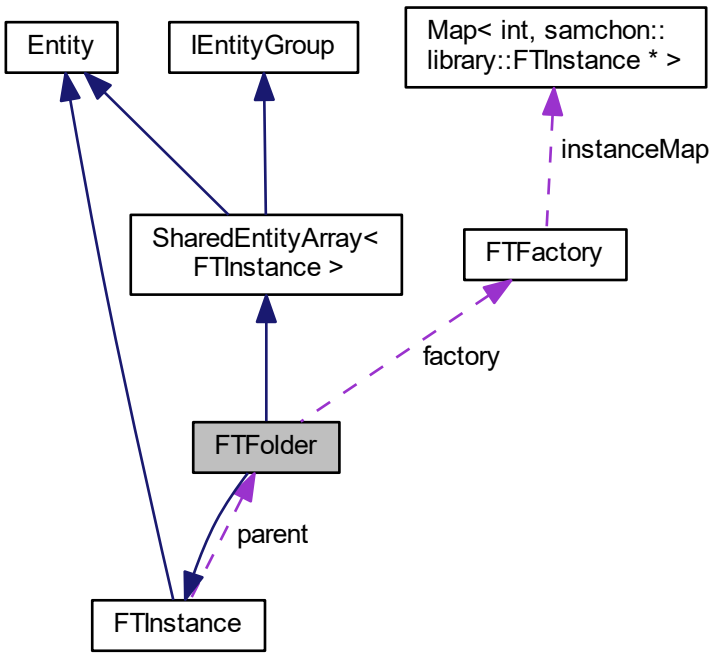
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTFile.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTFile.cpp

6.53 FTFolder Class Reference

A folder.

```
#include <FTFolder.hpp>
```

Collaboration diagram for FTFolder:



Public Member Functions

- FTFolder (FTFactory ∗, FTFolder ∗)

  Construct from factory and parent folder.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the Entity.

Protected Member Functions

- virtual auto CHILD_TAG () const -> std::string override

  A tag name of children.

Protected Attributes

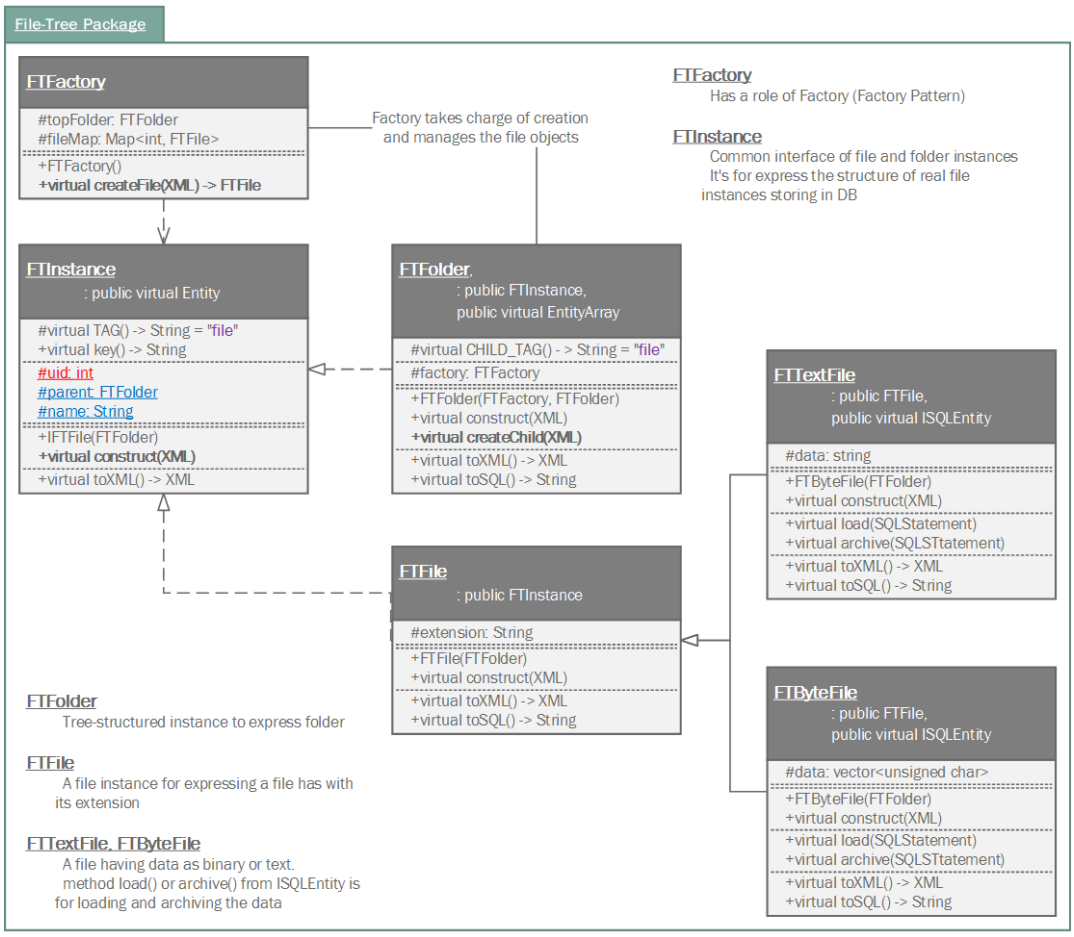
- FTFactory ∗ factory

  Factory instance for creating sub files.

6.53.1 Detailed Description

A folder.

A folder object has recursive and hierarchical relationship.



See also

samchon::library

Author

Jeongho Nam

6.53.2 Constructor & Destructor Documentation

6.53.2.1 FTFolder (FTFactory * *factory*, FTFolder * *parent*)

Construct from factory and parent folder.

Parameters

factory	Factory instance
parent	Parent folder that this folder is belonged to

6.53.3 Member Function Documentation

6.53.3.1 auto CHILD_TAG ( ) const -> std::string [override], [protected], [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

6.53.3.2 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from FTInstance.

The documentation for this class was generated from the following files:

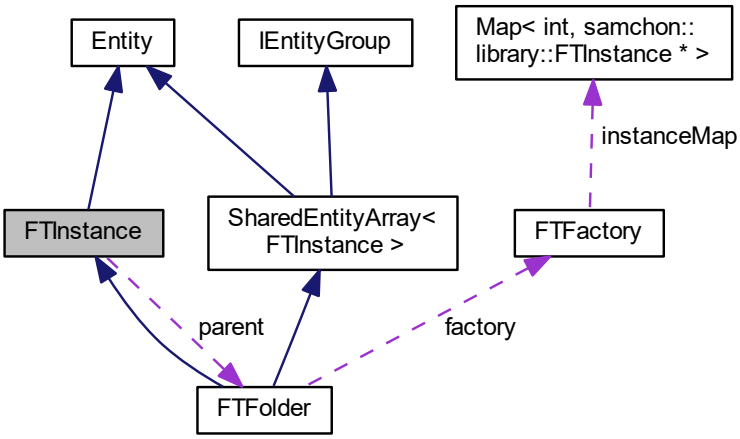
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTFolder.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTFolder.cpp

6.54 FTInstance Class Reference

An interface of file-tree.

`#include <FTInstance.hpp>`

Collaboration diagram for FTInstance:



Public Member Functions

- FTInstance (FTFolder *)

  Construct from parent folder.
- virtual auto key () const -> std::string

  Get a key that can identify the Entity uniquely.
- auto getUID () const -> int

  Get uid.
- auto getParent () const -> FTFolder *

  Get parent folder.
- auto getName () const -> std::string

  Get name.
- auto getComment () const -> std::string

  Get comment.
- virtual auto toXML () const -> std::shared_ptr< library::XML >

  Get an XML object represents the Entity.

Protected Member Functions

- virtual auto TAG () const -> std::string

  A tag name when represented by XML.

Protected Attributes

- FTFolder * parent

  Parent folder containing the instance.
- int uid

  Key, an unique id of file.
- std::string name

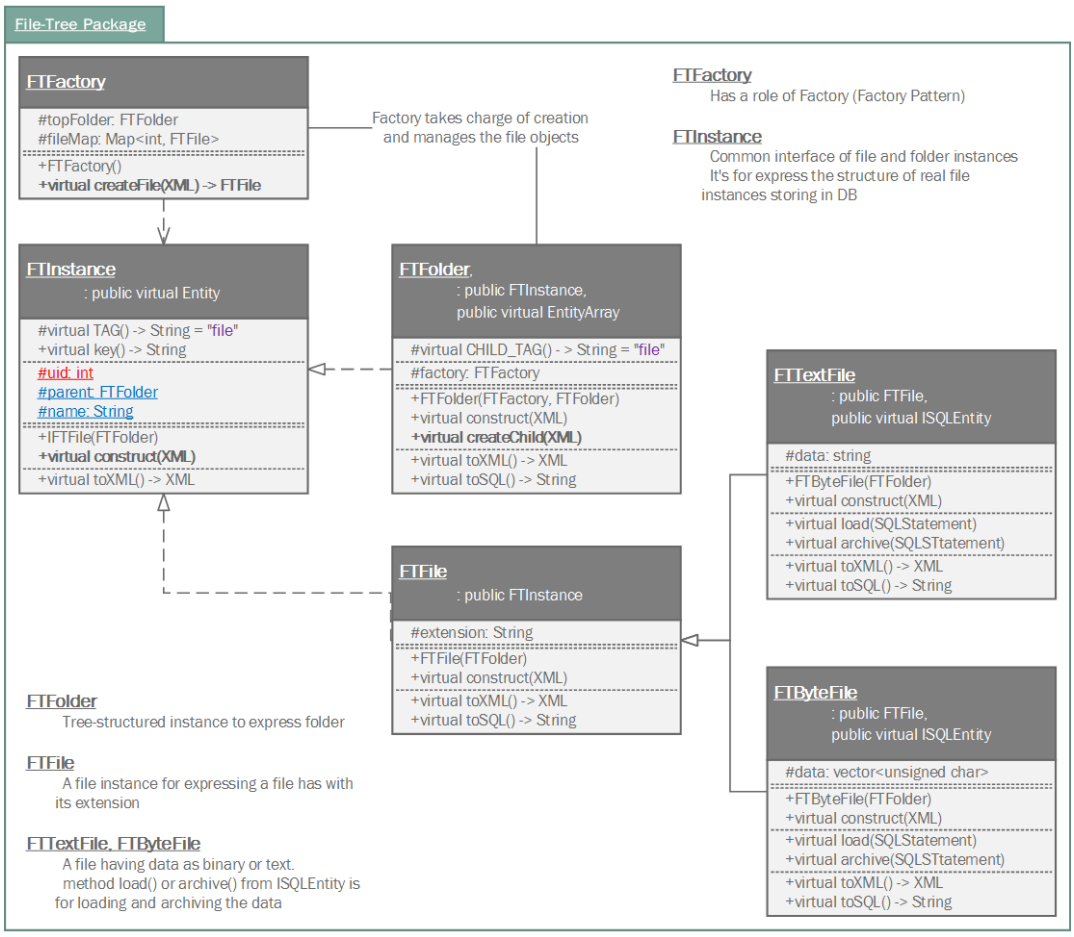
  Name of the file.
- std::string comment

  Comment of the file.

6.54.1 Detailed Description

An interface of file-tree.

An abstract class for folder and file instances.



See also

samchon::library

Author

Jeongho Nam

6.54.2 Constructor & Destructor Documentation

6.54.2.1 FTInstance (FTFolder * *parent*)

Construct from parent folder.

Parameters

parent	Parent folder that this instance is belonged to

6.54.3 Member Function Documentation

6.54.3.1 auto TAG ( ) const -> std::string [protected], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.54.3.2 auto key ( ) const -> std::string [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented from Entity.

6.54.3.3 auto toXML ( ) const -> std::shared_ptr<library::XML> [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

Reimplemented in NTFile, FTFolder, and FTFile.

The documentation for this class was generated from the following files:

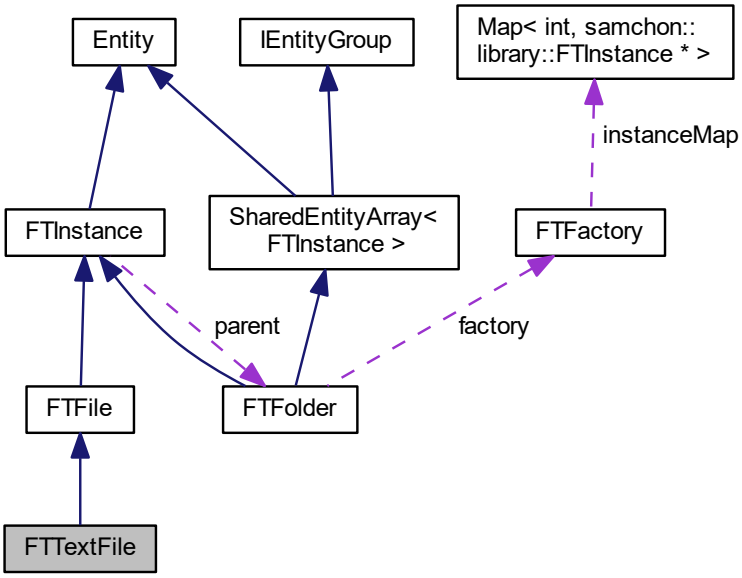
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTInstance.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTInstance.cpp

6.55 FTTextFile Class Reference

A text file.

```
#include <FTTextFile.hpp>
```

Collaboration diagram for FTTextFile:



Public Member Functions

- FTTextFile (FTFolder ∗)

  Construct from parent folder.
- auto getData () const -> std::string

  Get string content.

Protected Attributes

- std::string data

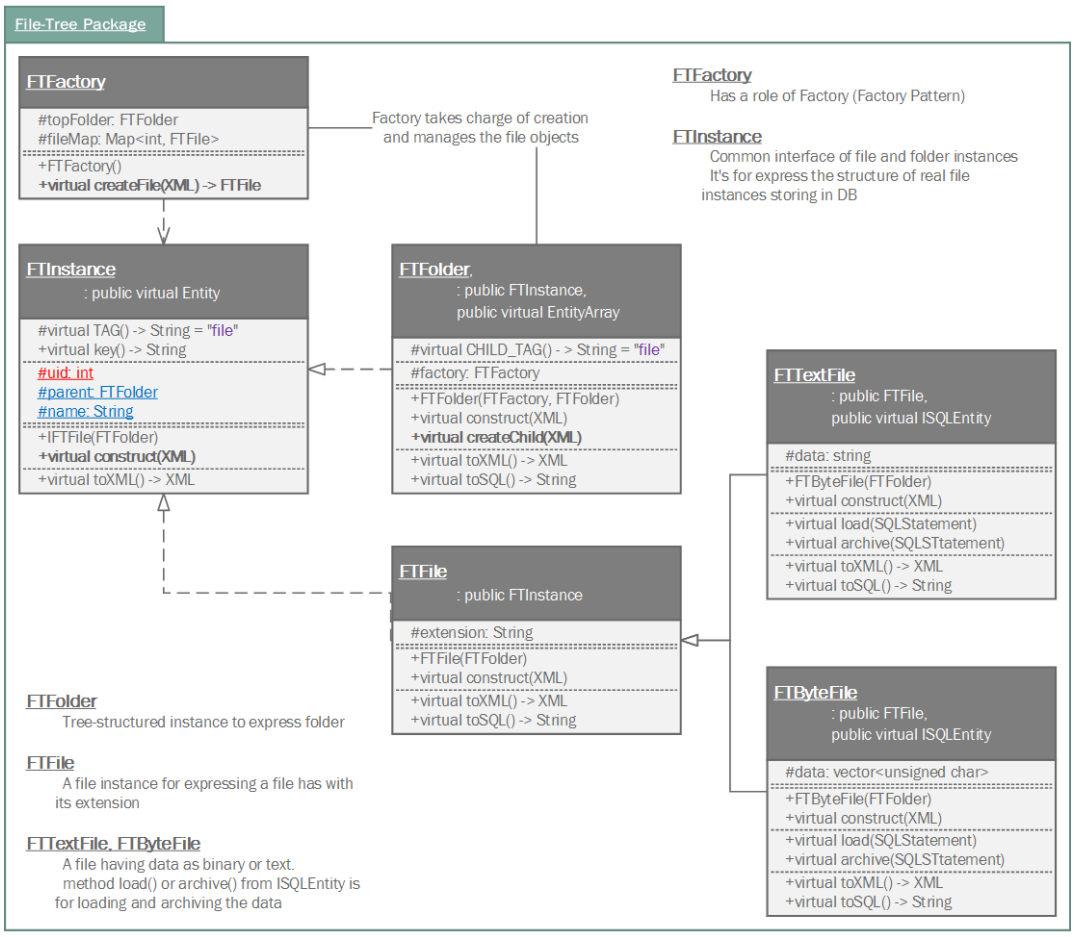
  Text recorded in the file.

Additional Inherited Members

6.55.1 Detailed Description

A text file.

A file having data as text.



See also

samchon::library

Author

Jeongho Nam

6.55.2 Constructor & Destructor Documentation

6.55.2.1 FTTextFile (FTFolder *)

Construct from parent folder.

Parameters

folder	Belonged folder

The documentation for this class was generated from the following file:

– D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/FTTextFile.hpp

6.56 GAParameters Struct Reference

A structure containing parameters of genetic algorithm.

```
#include <GAParameters.hpp>
```

Public Attributes

- size_t generation

  A number of generation of evolution.
- size_t population

  A number of population in a generation.
- size_t tournament

  A number of tournament in a selection.
- double mutationRate

  Rate of mutate ocurrence.

6.56.1 Detailed Description

A structure containing parameters of genetic algorithm.

The number and rate parameters in the structure affects success of evolution of genetic algorithm.

See also

library::GeneticAlgorithm
library::GAPopulation

Author

Jeongho Nam

6.56.2 Member Data Documentation

6.56.2.1 double mutationRate

Rate of mutate ocurrence.

A mutation rate determines the percentage of occurence of mutation in a sequence list.

- Value from 0.0 to 1.0

Note

- When mutationRate is too high, it is hard to ancitipate studying on genetic algorithm.
- When mutationRate is too low and initial set of genes(GeneArray) is far away from optimal, the evolution tends to wandering outside of he optimal.

The documentation for this struct was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/GAParameters.hpp

6.57 GAParameters Struct Reference

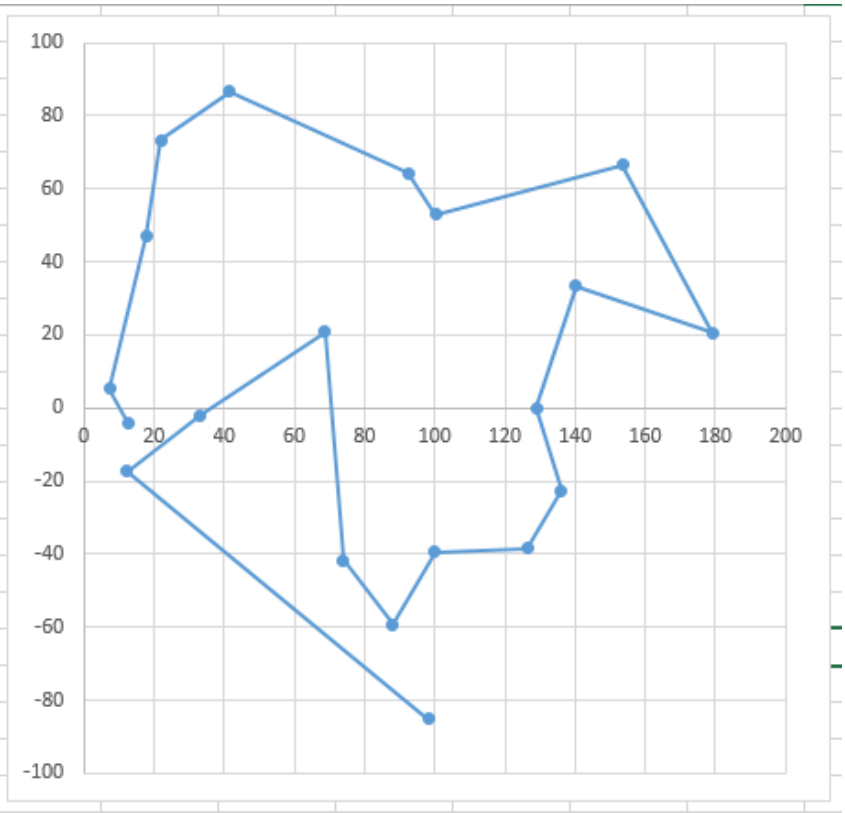
Parameters for Genetic-Algorithm.

```
#include <Scheduler.hpp>
```

6.57.1 Detailed Description

Parameters for Genetic-Algorithm.

Distance: 64,896.70 km		
uid	longitude	latitude
18	12.5233	-3.95529
15	7.23275	5.18342
7	17.7528	46.8665
2	22.0434	73.4133
20	41.6615	86.5372
13	92.427	64.3369
1	100.345	53.1067
9	153.521	66.5119
3	179.118	20.5073
10	140.113	33.3537
17	128.885	0.054607
4	136.012	-22.4915
6	126.352	-38.5031
16	99.9775	-39.3696
11	87.8859	-59.3663
14	73.8477	-41.5867
5	68.6563	20.8528
12	33.2162	-1.92823
8	12.1839	-17.3552
19	98.0967	-85.195



See also

samchon::library::GeneticAlgorithm

Author

Jeongho Nam

The documentation for this struct was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/tsp/Scheduler.hpp

6.58 GAPopulation< GeneArray, Compare > Class Template Reference

A population of a generation in G.A.

```
#include <GAPopulation.hpp>
```

Public Member Functions

- GAPopulation (std::shared_ptr< GeneArray > geneArray, size_t size)

  Construct from a GeneArray and size of the population.
- auto fitTest () const -> std::shared_ptr< GeneArray >

  Test fitness of each GeneArray in the population.

Private Member Functions

- GAPopulation (size_t size)
  Private Constructor with population.

Private Attributes

- std::vector< std::shared_ptr< GeneArray > > children
  Genes representing the population.

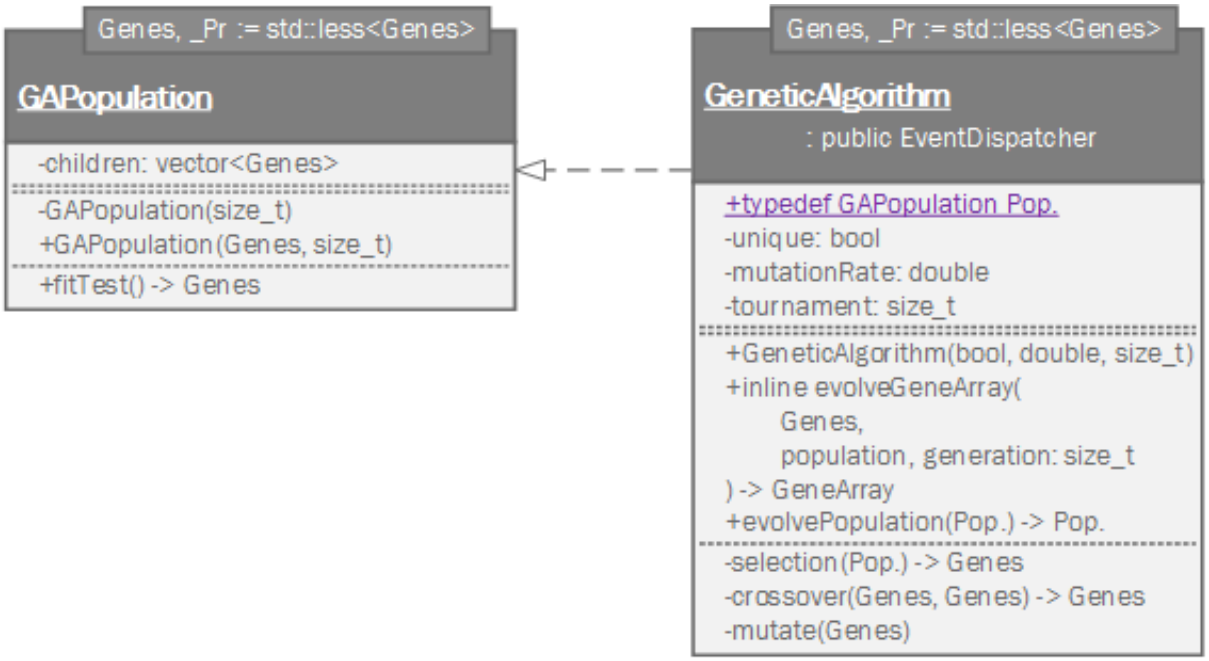
6.58.1 Detailed Description

template<typename GeneArray, typename Compare = std::less<GeneArray>>class samchon::library::GAPopulation< GeneArray, Compare >

A population of a generation in G.A.

GAPopulation is a class representing population of candidate genes(sequence listing) having an array of GeneArray as a member. GAPopulation also manages initial set of genes and handles fitting test direclty by the method *fitTest*.

The success of evolution of genetic alogirhtm is depend on the GAPopulation's initial set and fitting test. (GeneArray and Compare.)



Template Parameters

GeneArray	

An array(std::vector) containing genes as elments; sequnce listing

The GeneArray must be a type of *std::vector*.

Template Parameters

Compare	A comparison class (or struct) returns whether left gene is more optimal.
	
	Default template parameter of Compare is *std::less<GeneArray>*. It means to compare two std::vector (GeneArray must be a std::vector). Thus, you've to keep follwing rules.
	
	If you don't want to follow the rules or want a custom comparison class, you have to realize a comparison class. The following code is an example realizing the comparison class.
	
	`- GeneArray is inherited from <i>std::vector</i>`
	`- GeneArray has custom <i>auto operator<(const GeneArray &) const -> bool</i>`

```
template <typename _Ty>
struct MyCompare
{
auto operator()(const _Ty &newObj, const _Ty &prevObj) const -> bool;
};
```

Warning

Be careful for the mistakes of direction or position of Compare.

Most of logical errors failed to access optimal solution are occured by those mistakens.

See also

library::GeneticAlgorithm
samchon::library

Author

Jeonghoh Nam

6.58.2 Constructor & Destructor Documentation

6.58.2.1 GAPopulation (size_t *size*) [inline],[private]

Private Constructor with population.

Private constructor of GAPopulation does not create children (candidate genes) but fills only *nullptr* repeatedly following the size.

The private constructor is only for and handled by GeneticAlgorithm.

Parameters

size	Size of the population

6.58.2.2 GAPopulation (std::shared_ptr< GeneArray > *geneArray*, size_t *size*) [inline]

Construct from a GeneArray and size of the population.

Constructor with initial set of genes and size of the population.

The public constructor creates GeneArray(s) as population (size) having shuffled genes which are came from the initial set of genes (geneArray).

Note

The success of genetic algorithm is depend on the initial genes(geneArray).

Parameters

geneArray	An initial sequence listing
size	The size of population to have as children

6.58.3 Member Function Documentation

6.58.3.1 auto fitTest ( ) const -> std::shared_ptr<GeneArray> [inline]

Test fitness of each GeneArray in the population.

Returns

The best GeneArray in the population

The documentation for this class was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/GAPopulation.hpp

6.59 GeneticAlgorithm< GeneArray, Compare > Class Template Reference

A genetic algorithm class.

```
#include <GeneticAlgorithm.hpp>
```

Public Member Functions

- GeneticAlgorithm (bool unique, double mutationRate=0.015, size_t tournament=10)
  Construct from parameters of Genetic Algorithm.
- auto evolveGeneArray (std::shared_ptr< GeneArray > geneArray, size_t population, size_t generation) const -> std::shared_ptr< GeneArray >
  Evolve a GeneArray.
- auto evolvePopulation (std::shared_ptr< MyPopulation > population) const -> std::shared_ptr< My↩Population >
  Evolve population, a mass of GeneArray(es)

Private Member Functions

- auto selection (std::shared_ptr< MyPopulation > population) const -> std::shared_ptr< GeneArray >
  Select the best GeneArray in population from tournament.
- auto crossover (std::shared_ptr< GeneArray > &parent1, std::shared_ptr< GeneArray > &parent2) const -> std::shared_ptr< GeneArray >
  Create a new GeneArray by crossing over two GeneArray(s)
- void mutate (std::shared_ptr< GeneArray > geneArray) const
  Cause a mutation on the GeneArray.

Private Attributes

- bool unique
  Whether each element (Gene) is unique in their GeneArray.
- double mutationRate
  Rate of mutate ocurrence.
- size_t tournament
  Size of tournament in selection.

6.59.1 Detailed Description

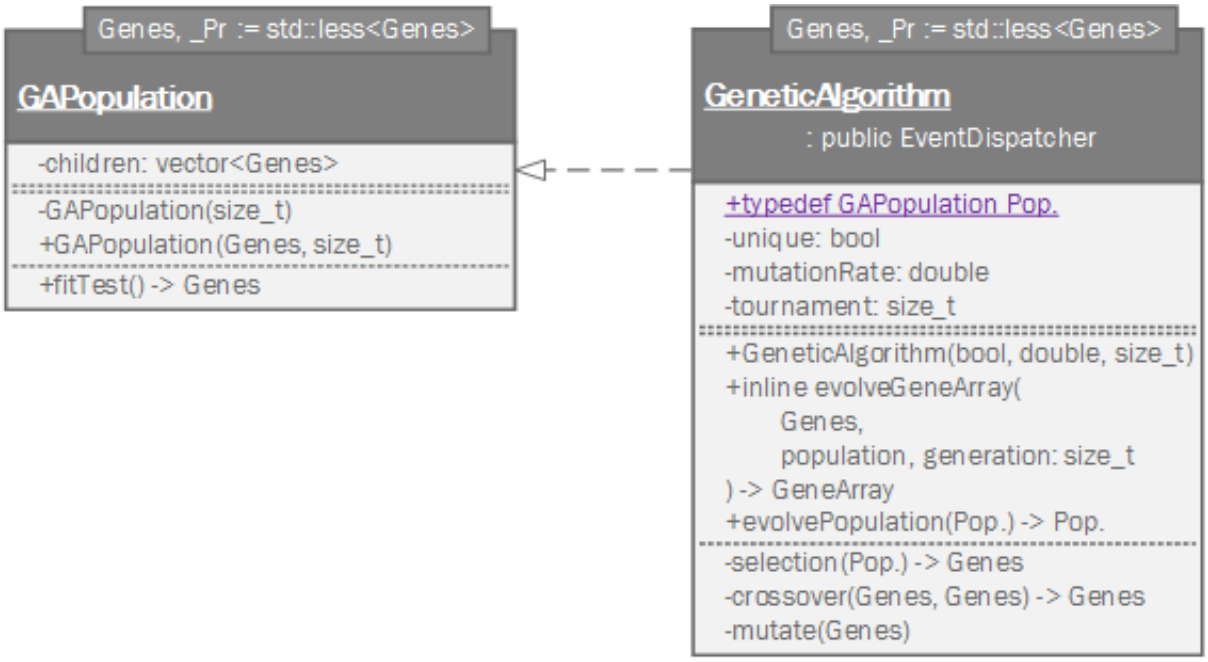
template<typename GeneArray, typename Compare = std::less<GeneArray>>class samchon::library::GeneticAlgorithm< GeneArray, Compare >

A genetic algorithm class.

In the field of artificial intelligence, a genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search problems.

Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

– Referenced Wikipedia: https://en.wikipedia.org/wiki/Genetic_algorithm



Template Parameters

GeneArray	An array(std::vector) containing genes as elments; sequnce listing. The GeneArray must be a type of *std::vector*.
Compare	A comparison class (or struct) returns whether left gene is more optimal. Default template parameter of Compare is *std::less<GeneArray>*. It means to compare two std::vector (GeneArray must be a std::vector). Thus, you've to keep follwing rules. If you don't want to follow the rules or want a custom comparison class, you have to realize a comparison class. The following code is an example realizing the comparison class. - GeneArray is inherited from std::vector - GeneArray has custom auto operator<(const GeneArray &) const -> bool

```
template <typename _Ty>
struct MyCompare
{
auto operator()(const _Ty &newObj, const _Ty &prevObj) const -> bool;
};
```

Example source

TSP - Traveling Salesman Problem

examples/tsp/Scheduler.hpp

```
1 #pragma once
2 #include <vector>
3 #include <memory>
4
5 namespace samchon
6 {
7     namespace example
8     {
9         namespace tsp
10         {
11             class Travel;
12
23             struct GAParameters
24             {
25                 double mutationRate;
26                 size_t tornament;
27
28                 size_t population;
29                 size_t generation;
30             };
31
38             class Scheduler
39             {
40             private:
44                 std::shared_ptr<Travel> travel;
45
49                 struct GAParameters gaParameters;
50
51             public:
55                 Scheduler(std::shared_ptr<Travel>, const struct GAParameters &);
56
64                 auto optimize() -> std::shared_ptr<Travel>;
65             };
66         };
67     };
68 };
```

examples/tsp/Travel.hpp

```
1 #pragma once
2 #include <vector>
3 #include <string>
4
5 namespace samchon
6 {
7     namespace example
8     {
9         namespace tsp
10         {
11             class GeometryPoint;
12
24             class Travel
25                 : public std::vector<GeometryPoint*>
26             {
27             private:
28                 typedef std::vector<GeometryPoint*> super;
29
36                 double distance;
37
38             public:
39                 /* ------------------------------------------------------------
40                     CONSTRUCTORS
41                 ------------------------------------------------------------ */
45                 Travel();
46
55                 Travel(const Travel&);
56
60                 Travel(Travel&&);
61
62             private:
63                 /* ------------------------------------------------------------
64                     CALCULATORS
65                 ------------------------------------------------------------ */
69                 auto calcDistance() const -> double;
70
71             public:
79                 auto operator<(const Travel &) const -> bool;
80
81                 /* ------------------------------------------------------------
```

```
                    EXPORTER
                ------------------------------------------------------------ */
                auto toString() const -> std::string;
            };
        };
    };
};
```

examples/tsp/GeometryPoint.hpp

```
#pragma once
#include <string>

namespace samchon
{
    namespace example
    {
        namespace tsp
        {
            class GeometryPoint
            {
            private:
                int uid;

                double longitude;

                double latitude;

            public:
                /* ------------------------------------------------------------
                    CONSTRUCTORS
                ------------------------------------------------------------ */
                GeometryPoint(int);

                GeometryPoint(int, double, double);

                /* ------------------------------------------------------------
                    CALCULATOR
                ------------------------------------------------------------ */
                auto calcDistance(const GeometryPoint &) const -> double;

                /* ------------------------------------------------------------
                    EXPORTER
                ------------------------------------------------------------ */
                auto toString() const -> std::string;
            };
        };
    };
};
```

examples/tsp/main.cpp

```
#include <iostream>
#include <Windows.h>

#include "Scheduler.hpp"
#include "Travel.hpp"
#include "GeometryPoint.hpp"

using namespace std;
using namespace samchon::example::tsp;

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

void toClipboard(const string &);

void main()
{
    //GEOMETRY COORPORATES
    shared_ptr<Travel> travel(new Travel());
    for(int i = 0; i < 20; i++)
        travel->push_back(new GeometryPoint(i + 1));

    //OPTIMIZING
```

```
    struct GAParameters gaParameters = {.03, 50, 100, 300};

    Scheduler scheduler(travel, gaParameters);
    travel = scheduler.optimize();

    //PRINTING
    string &str = travel->toString();
    toClipboard(str);
    cout << str << endl;

    system("pause");
}

void toClipboard(const string &str)
{
    OpenClipboard(0);
    EmptyClipboard();
    HGLOBAL hg = GlobalAlloc(GMEM_MOVEABLE, str.size());

    if (!hg)
    {
        CloseClipboard();
        return;
    }
    memcpy(GlobalLock(hg), str.c_str(), str.size());

    GlobalUnlock(hg);
    SetClipboardData(CF_TEXT, hg);
    CloseClipboard();
    GlobalFree(hg);
}
```

examples/tsp/Scheduler.cpp

```
#include "Scheduler.hpp"
#   include "Travel.hpp"

#include <samchon/library/GeneticAlgorithm.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::example::tsp;

Scheduler::Scheduler(shared_ptr<Travel> travel, const GAParameters &gaParameteres)
{
    this->travel = travel;
    this->gaParameters = gaParameteres;
}
auto Scheduler::optimize() -> shared_ptr<Travel>
{
    GeneticAlgorithm<Travel> geneticAlgorithm
    (
        true,
        gaParameters.mutationRate,
        gaParameters.tornament
    );

    travel =
        geneticAlgorithm.evolveGeneArray
        (
            travel,
            gaParameters.population,
            gaParameters.generation
        );
    return travel;
}
```

examples/tsp/Travel.cpp

```
#include "Travel.hpp"
#   include "GeometryPoint.hpp"

#include <samchon/library/StringUtil.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::example::tsp;

/* ----------------------------------------------------------
    CONSTRUCTORS
---------------------------------------------------------- */
Travel::Travel()
    : super()
{
    distance = INT_MIN;
}
```

```
Travel::Travel(const Travel &travel)
    : super(travel)
{
    distance = INT_MIN;
}
Travel::Travel(Travel &&travel)
    : super(move(travel))
{
    distance = travel.distance;
}

/* ------------------------------------------------------------
    CALCULATORS
------------------------------------------------------------ */
auto Travel::calcDistance() const -> double
{
    if(this->distance != INT_MIN)
        return this->distance;

    double distance = 0.0;
    for(size_t i = 1; i < size(); i++)
        distance += at(i-1)->calcDistance(*at(i));

    ((Travel*)this)->distance = distance;
    return distance;
}
auto Travel::operator<(const Travel &travel) const -> bool
{
    return this->calcDistance() < travel.calcDistance();
}

/* ------------------------------------------------------------
    EXPORTER
------------------------------------------------------------ */
auto Travel::toString() const -> string
{
    string str =
        "Distance: " + StringUtil::numberFormat(calcDistance()) + " km\n" +
        "uid    longitude   latitude\n";

    for(size_t i = 0; i < size(); i++)
        str += at(i)->toString() + "\n";

    return move(str);
}
```

examples/tsp/GeometryPoint.cpp

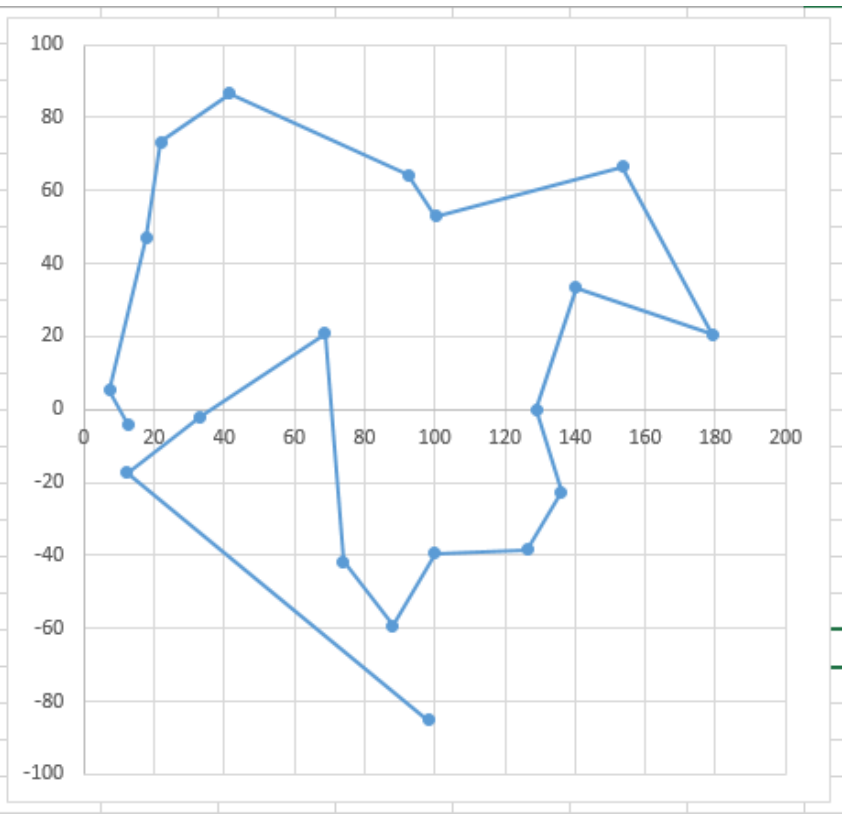
```
#include "GeometryPoint.hpp"

#include <random>
#include <cmath>
#include <samchon/library/Math.hpp>
#include <samchon/library/StringUtil.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::example::tsp;

/* ------------------------------------------------------------
    CONSTRUCTORS
------------------------------------------------------------ */
GeometryPoint::GeometryPoint(int uid)
{
    this->uid = uid;
    this->longitude = Math::random() * 180.0;
    this->latitude = Math::random() * 180 - 90.0;
}
GeometryPoint::GeometryPoint(int uid, double longitude, double latitude)
{
    this->uid = uid;
    this->longitude = longitude;
    this->latitude = latitude;
}

/* ------------------------------------------------------------
    CALCULATOR
------------------------------------------------------------ */
auto GeometryPoint::calcDistance(const GeometryPoint &point) const -> double
{
    if (longitude == point.longitude && latitude == point.latitude)
        return 0.0;

    double latitude_radian1 = Math::degree_to_radian(this->latitude);
    double latitude_radian2 = Math::degree_to_radian(point.latitude);
    double theta = this->longitude - point.longitude;

```

```
    double val =
        sin(latitude_radian1) * sin(latitude_radian2)
        + cos(latitude_radian1) * cos(latitude_radian2) * cos(
    Math::degree_to_radian(theta));

    val = acos(val);
    val = Math::radian_to_degree(val);
    val = val * 60 * 1.1515;
    val = val * 1.609344;

    return val;
}

/* ------------------------------------------------------------
    EXPORTER
------------------------------------------------------------ */
auto GeometryPoint::toString() const -> string
{
    return StringUtil::substitute
        (
            "{1}\t{2}\t{3}",
            uid, longitude, latitude
        );
}
```

Result of the example

Distance: 64,896.70 km

uid	longitude	latitude
18	12.5233	-3.95529
15	7.23275	5.18342
7	17.7528	46.8665
2	22.0434	73.4133
20	41.6615	86.5372
13	92.427	64.3369
1	100.345	53.1067
9	153.521	66.5119
3	179.118	20.5073
10	140.113	33.3537
17	128.885	0.054607
4	136.012	-22.4915
6	126.352	-38.5031
16	99.9775	-39.3696
11	87.8859	-59.3663
14	73.8477	-41.5867
5	68.6563	20.8528
12	33.2162	-1.92823
8	12.1839	-17.3552
19	98.0967	-85.195



Warning

Be careful for the mistakes of direction or position of Compare.

Most of logical errors failed to access optimal solution are occured by those mistakens.

See also

library::GAPopulation
samchon::library

Author

Jeongho Nam

6.59.2 Constructor & Destructor Documentation

6.59.2.1 GeneticAlgorithm (bool *unique*, double *mutationRate* = 0.015, size_t *tournament* = 10) [inline]

Construct from parameters of Genetic Algorithm.

Parameters

unique	Whether each Gene is unique in their GeneArray
mutationRate	Rate of mutation
tournament	Size of tournament in selection
elitism	Whether to keep the elitest GeneArray

6.59.3 Member Function Documentation

6.59.3.1 auto evolveGeneArray (std::shared_ptr< GeneArray > *geneArray*, size_t *population*, size_t *generation*) const -> std::shared_ptr<GeneArray> [inline]

Evolve a GeneArray.

Convinient method accessing to evolvePopulation().

Parameters

geneArray	An initial set of genes; sequence listing
population	Size of population in a generation
generation	Size of generation in evolution

Returns

A evolved GeneArray optimally

6.59.3.2 auto evolvePopulation (std::shared_ptr< MyPopulation > *population*) const -> std::shared_ptr<MyPopulation> [inline]

Evolve population, a mass of GeneArray(es)

Parameters

population	An initial population

6.59.3.3 auto selection (std::shared_ptr< MyPopulation > *population*) const -> std::shared_ptr<GeneArray> [inline],[private]

Select the best GeneArray in population from tournament.

Selection is the stage of a genetic algorithm in which individual genomes are chosen from a population for later breeding (using crossover operator). A generic selection procedure may be implemented as follows:

1. The fitness function is evaluated for each individual, providing fitness values, which are then normalized. Normalization means dividing the fitness value of each individual by the sum of all fitness values, so that the sum of all resulting fitness values equals 1.
2. The population is sorted by descending fitness values.
3. Accumulated normalized fitness values are computed (the accumulated fitness value of an individual is the sum of its own fitness value plus the fitness values of all the previous individuals). The accumulated fitness of the last individual should be 1 (otherwise something went wrong in the normalization step).

4. A random number R between 0 and 1 is chosen.
5. The selected individual is the first one whose accumulated normalized value is greater than R.

– Referenced Wekipedia: https://en.wikipedia.org/wiki/Selection_(genetic_algorithm)

Parameters

population	The target of tournament

Returns

The best genes derived by the tournament

6.59.3.4 auto crossover (std::shared_ptr< GeneArray > & *parent1*, std::shared_ptr< GeneArray > & *parent2*) const -> std::shared_ptr<GeneArray> [inline],[private]

Create a new GeneArray by crossing over two GeneArray(s)

crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based.

Cross over is a process of taking more than one parent solutions and producing a child solution from them. There are methods for selection of the chromosomes.

– Referenced Wikipedia: https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

Parameters

parent1	A parent sequence listing
parent2	A parent sequence listing

6.59.3.5 void mutate (std::shared_ptr< GeneArray > *geneArray*) const [inline],[private]

Cause a mutation on the GeneArray.

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation.

Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better solution by using mutation.

Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search.

– Referenced Wikipedia: https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)

Note

Muttion is pursuing diversity. Mutation is useful for avoiding the following problem.

When initial set of genes(GeneArray) is far away from optimail, without mutation (only with selection and crossover), the genetic algorithm has a tend to wandering outside of the optimal.

Genes in the GeneArray will be swapped following percentage of the mutationRate.

Parameters

geneArray	A container of genes to mutate

See also

mutationRate;

6.59.4 Member Data Documentation

6.59.4.1 double mutationRate [private]

Rate of mutate ocurrence.

The mutationRate determines the percentage of occurence of mutation in a GeneArray.

Note

- When mutationRate is too high, it is hard to ancitipate studying on genetic algorithm.
- When mutationRate is too low and initial set of genes(GeneArray) is far away from optimal, the evolution tends to wandering outside of he optimal.

The documentation for this class was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/GeneticAlgorithm.hpp

6.60 GeometryPoint Class Reference

A geometry coordinates (point)

```
#include <GeometryPoint.hpp>
```

Public Member Functions

- GeometryPoint (int)

  Construct from uid.
- GeometryPoint (int, double, double)

  Construct from uid and geometry coordinates.
- auto calcDistance (const GeometryPoint &) const -> double

  Calculate distance between target Branch.
- auto toString () const -> std::string

  Convert the Branch to String.

Private Attributes

- int uid

  An unique id for uniqueness.
- double longitude

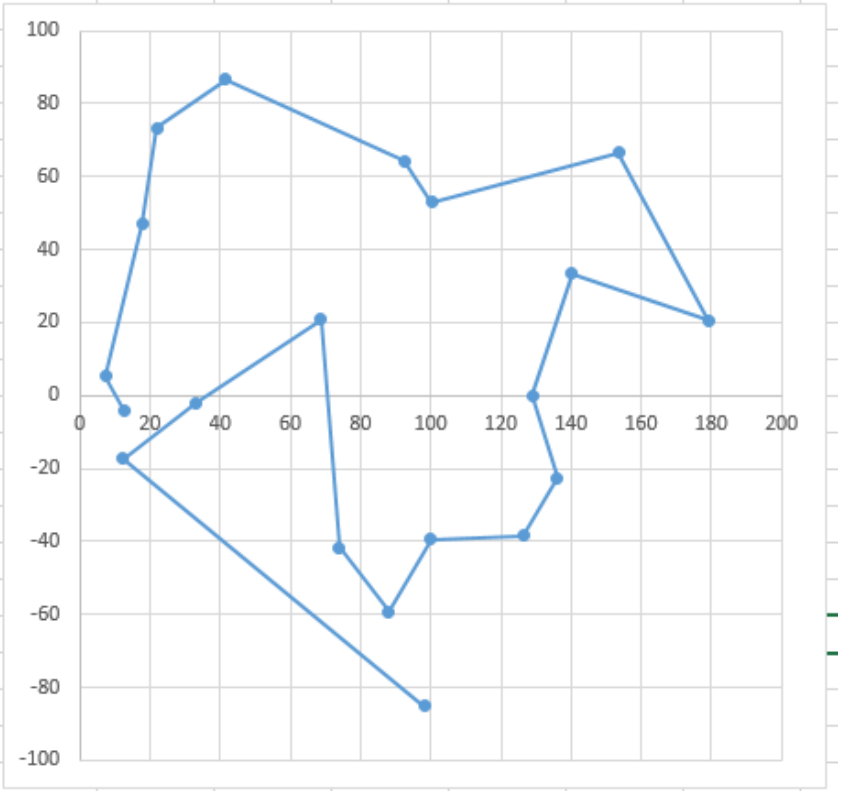
  The longitude; coordinates X.
- double latitude

  The latitude, coordinates Y.

6.60.1 Detailed Description

A geometry coordinates (point)

Distance: 64,896.70 km		
uid	longitude	latitude
18	12.5233	-3.95529
15	7.23275	5.18342
7	17.7528	46.8665
2	22.0434	73.4133
20	41.6615	86.5372
13	92.427	64.3369
1	100.345	53.1067
9	153.521	66.5119
3	179.118	20.5073
10	140.113	33.3537
17	128.885	0.054607
4	136.012	-22.4915
6	126.352	-38.5031
16	99.9775	-39.3696
11	87.8859	-59.3663
14	73.8477	-41.5867
5	68.6563	20.8528
12	33.2162	-1.92823
8	12.1839	-17.3552
19	98.0967	-85.195



Author

Jeongho Nam

6.60.2 Constructor & Destructor Documentation

6.60.2.1 GeometryPoint (int *uid*)

Construct from uid.

Geometry coordinates, longitude and latitude will have a random value between

- Longitude between 0 and 180.
- Latitude between -90 and 90.

6.60.2.2 GeometryPoint (int *uid,* double *longitude,* double *latitude*)

Construct from uid and geometry coordinates.

Parameters

longitude	Longitude, a coordinate of the geometry point
latitude	Latitude, a coordinate of the geometry point

6.60.3 Member Function Documentation

6.60.3.1 auto calcDistance (const GeometryPoint & *point*) const -> double

Calculate distance between target Branch.

Referenced from http://thesunrises.tistory.com/958 (Geometry points to killometers)

Parameters

branch	The target Branch to calculate

6.60.3.2 auto toString ( ) const -> std::string

Convert the Branch to String.

Have of form of tab and enter delimeters for Excel.

- `{$uid}  {$longitude}  {$latitude}`

Returns

A string represents the GeometryPoint

The documentation for this class was generated from the following files:

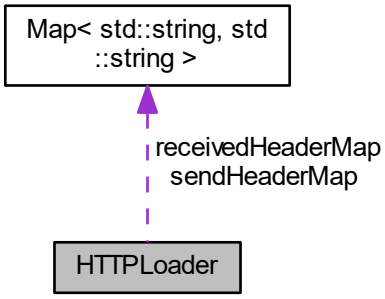
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/tsp/GeometryPoint.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/tsp/GeometryPoint.cpp

6.61 HTTPLoader Class Reference

A http, web-page loader.

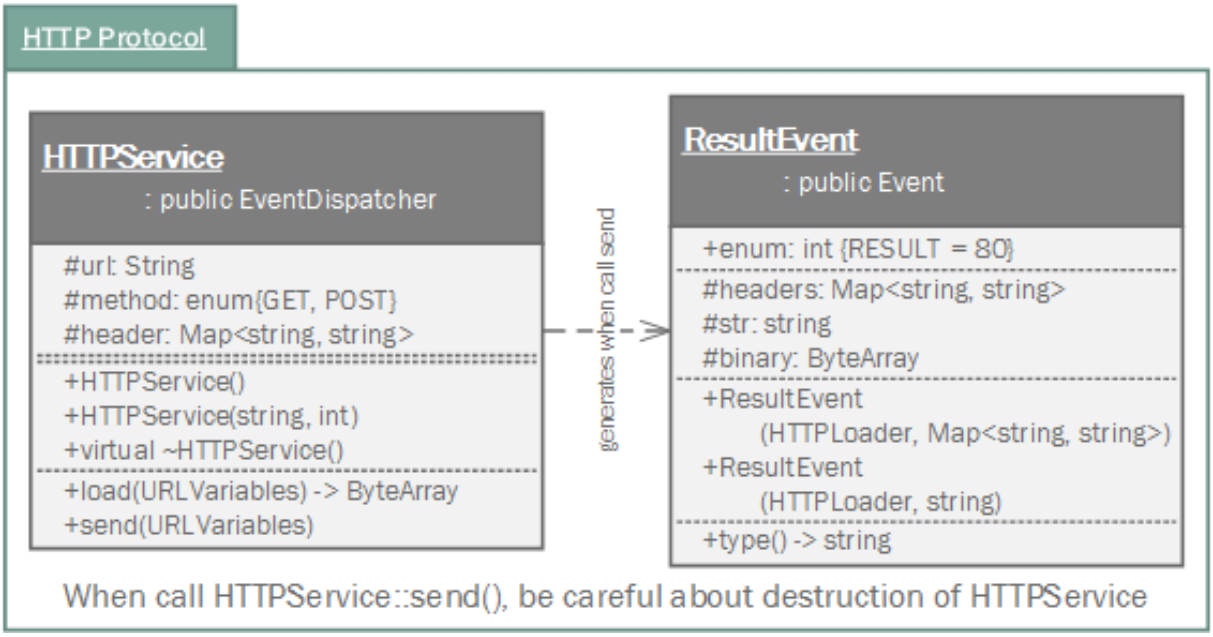
`#include <HTTPLoader.hpp>`

Collaboration diagram for HTTPLoader:



6.61.1 Detailed Description

A http, web-page loader.



Note

Depreciated

See also

samchon::library

Author

Jeongho Nam

The documentation for this class was generated from the following file:

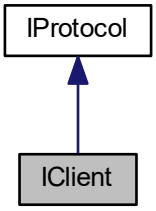
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/HTTPLoader.hpp

6.62 IClient Class Reference

An interface for a client.

`#include <IClient.hpp>`

Collaboration diagram for IClient:



Public Member Functions

- IClient ()

  Default Constructor.
- virtual void listen ()

  Listens message from a related system.
- virtual void sendData (std::shared_ptr< Invoke >)

  Sends message to a related system.

Protected Member Functions

- virtual auto BUFFER_SIZE () const -> size_t

  Buffer size of network I/O.
- virtual void _replyData (std::shared_ptr< Invoke >)

  A method for pre-processing replied Invoke message.

Protected Attributes

- Socket * socket

  Socket for network I/O.
- std::mutex * sendMtx

  A mutex for sending message.

6.62.1 Detailed Description

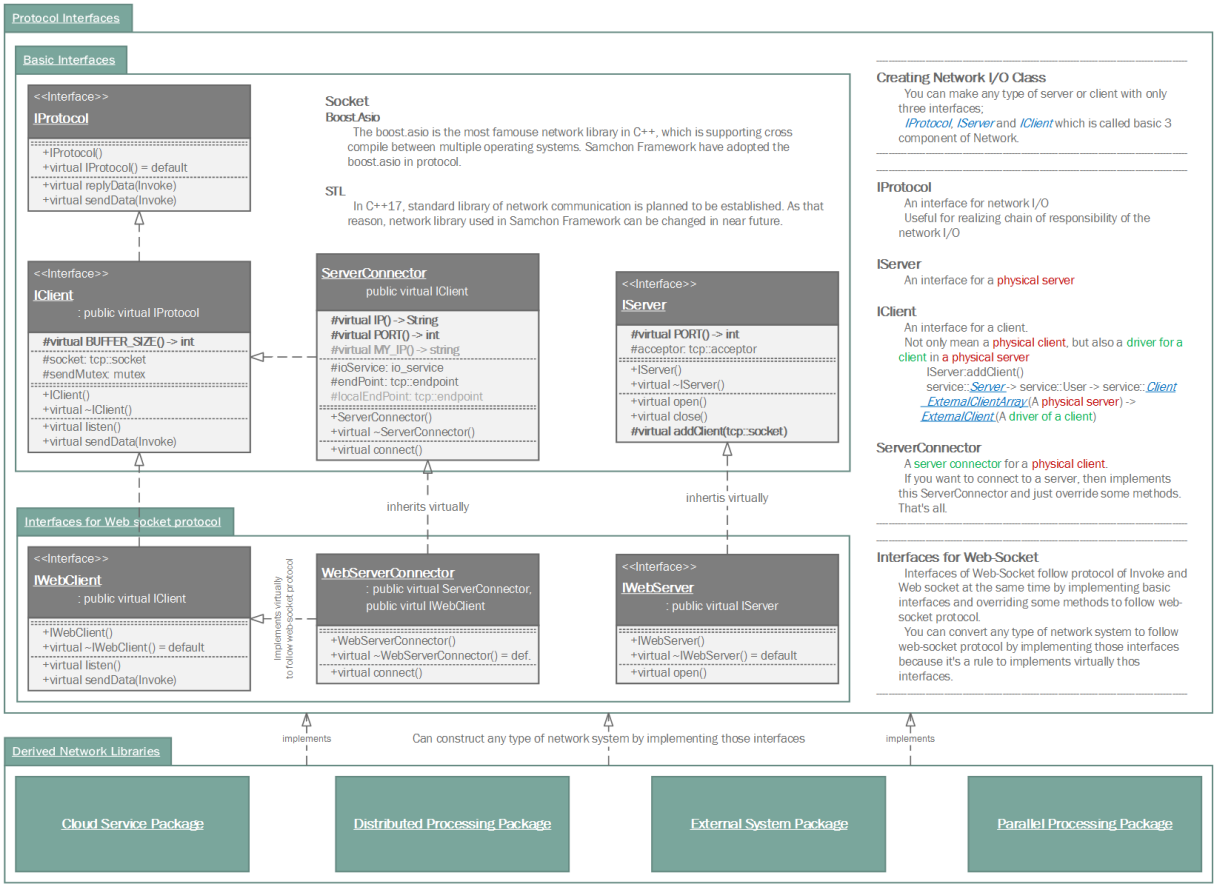
An interface for a client.

IClient is an interface not only for a physical client, but also a driver for a client in a physical server.

- IServer:addClient()
  - service::Server -> service::User -> service::Client
  - ExternalClientArray (A physical server) -> ExternalClient (A driver of a client)

IClient is one of the basic 3 + 1 components that can make any type of network system in Samchon Framework with IProtocol and IServer. Looking around classes in Samchon Framework, you can see all classes related with client are implemented from the IClient.

IClient takes a role of real network I/O. IClient sends data to otherside network system from IProtocol::send↩Data(). IClient also gets data from otherside network system, generates standard Invoke message from parsing the replied data and shift the Invoke message to IProtocol::replyData().



Author

Jeongho Nam

6.62.2 Member Function Documentation

6.62.2.1 void listen ( ) [virtual]

Listens message from a related system.

Note

It holds (monopolies) a thread.

Reimplemented in IWebClient.

6.62.2.2 void sendData (std::shared_ptr< Invoke >) [virtual]

Sends message to a related system.

Parameters

invoke	Invoke message to send

Reimplemented from IProtocol.

Reimplemented in DistributedSystem, Client, and IWebClient.

6.62.2.3 void _replyData (std::shared_ptr< Invoke >) [protected],[virtual]

A method for pre-processing replied Invoke message.

You can realize a pre-processing about the replied Invoke message by overriding the IClient::_replyData().

Archiving a history log, filtering and truncating, let handling the message with a new thread, calculating performance by estimating elapsed time about the Invoke message, and so on. Anything you can define pre-processing works before IProtocol::replyData().

- Archive history log -> cloud service
- Estimating performance -> distributed and parallel processing system

Parameters

invoke	An invoke message to be pre-processed and shifted to IProtocol::replyData().

Reimplemented in ParallelSystem, SlaveSystem, and ParallelSystem.

The documentation for this class was generated from the following files:

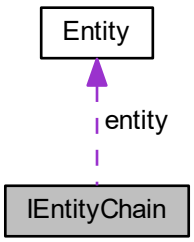
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IClient.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IClient.cpp

6.63 IEntityChain Class Reference

A chain of entity.

```
#include <IEntityChain.hpp>
```

Collaboration diagram for IEntityChain:



Public Member Functions

- IEntityChain (Entity *)
  Default Constructor.

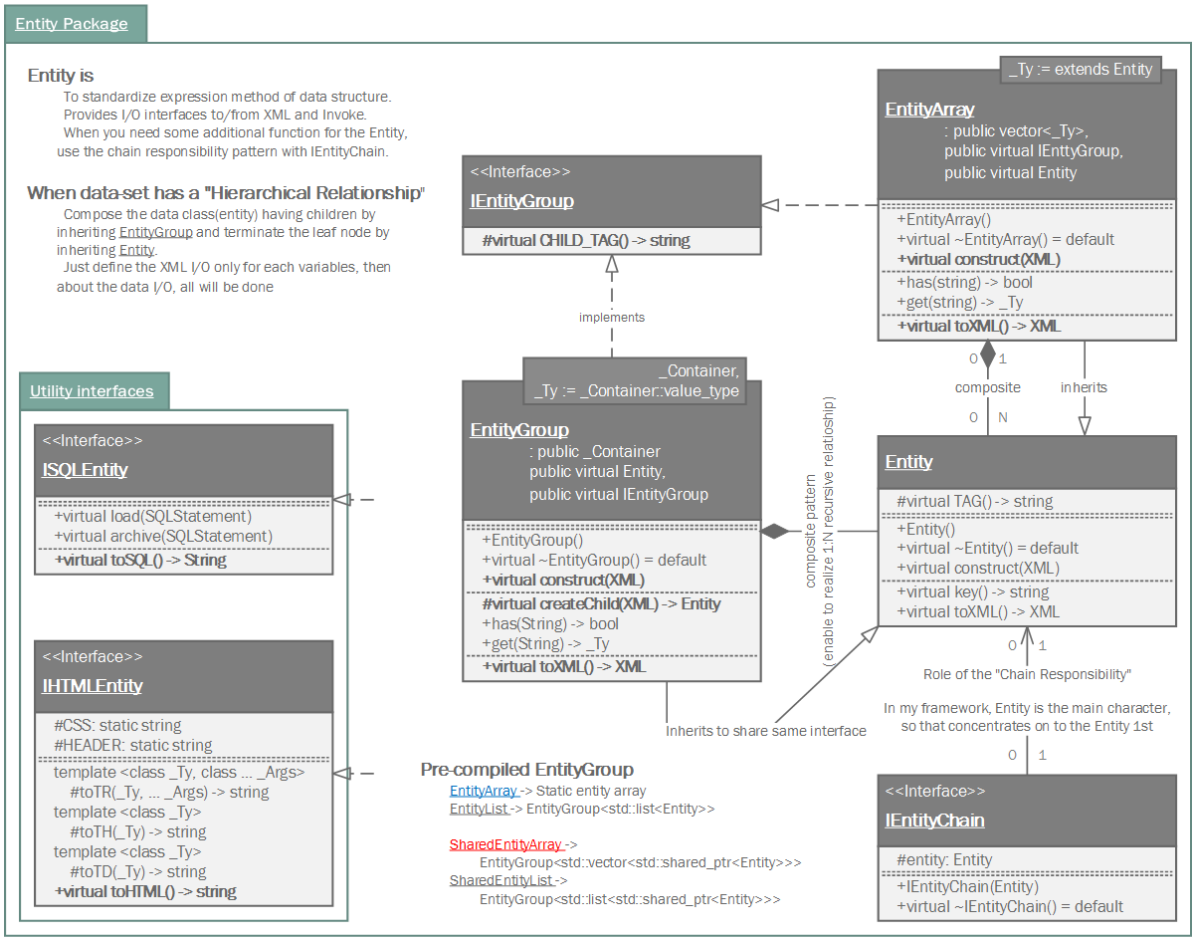
Protected Attributes

- Entity * entity
  An entity related with the chain.

6.63.1 Detailed Description

A chain of entity.

IEntityChain is an interface takes role of control about an entity by chain of responsibliy pattern. It's suitable for a supplementary processes are required on an entity like calculation or historical regression about an entity.



See also

samchon::protocol

Author

Jeongho Nam

6.63.2 Constructor & Destructor Documentation

6.63.2.1 IEntityChain (Entity * *entity*)

Default Constructor.

Parameters

entity	A related entity.

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IEntityChain.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IEntityChain.cpp

6.64 IEntityGroup Class Reference

An iternface for entity group.

`#include <IEntityGroup.hpp>`

Public Member Functions

- IEntityGroup ()
  Default Constructor.

Protected Member Functions

- virtual auto CHILD_TAG () const -> std::string=0
  A tag name of children.

6.64.1 Detailed Description

An iternface for entity group.

IEntityGroup is an interface for a container of entity. It does not have any special role or function. The IEntityGorup is only for discriminate an entity is single instance or gorup of entity.

- Group of entity is divided into EntityArray and EntityGroup.
  - Even they are template classes so that cannot be discriminated globally.
- dynamic_cast<IEntityGroup>(*some entity instance*)
  - If true then, it's not single entity instance, but group of entity.
  - False then, it's a single instance.

Author

Jeongho Nam

6.64.2 Member Function Documentation

6.64.2.1 virtual auto CHILD_TAG ( ) const -> std::string [protected], [pure virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implemented in Invoke, NTCriteria, ExternalSystemArray, ExternalSystem, NTParameter, ChatRoomArray, InvokeHistoryArray, NTParameterArray, and FTFolder.

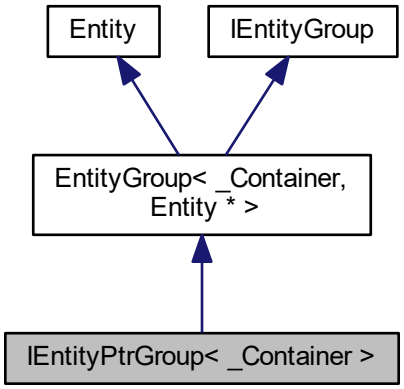
The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IEntityGroup.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IEntityGroup.cpp

6.65 IEntityPtrGroup< _Container > Class Template Reference

`#include <IEntityPtrGroup.hpp>`

Collaboration diagram for IEntityPtrGroup< _Container >:



Additional Inherited Members

6.65.1 Detailed Description

template<typename _Container>class samchon::protocol::IEntityPtrGroup< _Container >

EntityPtrGroup is an EntityGroup having children Entity as their pointers.

The documentation for this class was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IEntityPtrGroup.hpp

6.66 IHTMLEntity Class Reference

An interface supporting conversion to html.

`#include <IHTMLEntity.hpp>`

Public Member Functions

- IHTMLEntity ()
  Default Constructor.
- virtual auto toHTML () const -> std::string=0
  Get an html string represents the Entity.

Static Public Member Functions

- template<typename _Ty , typename... _Args>
  static auto toTH (const _Ty &val, const _Args &...args) -> std::string
  Get a string represents a <th> tag.

– template<typename _Ty , typename... _Args>
static auto toTR (const _Ty &val, const _Args &...args) -> std::string

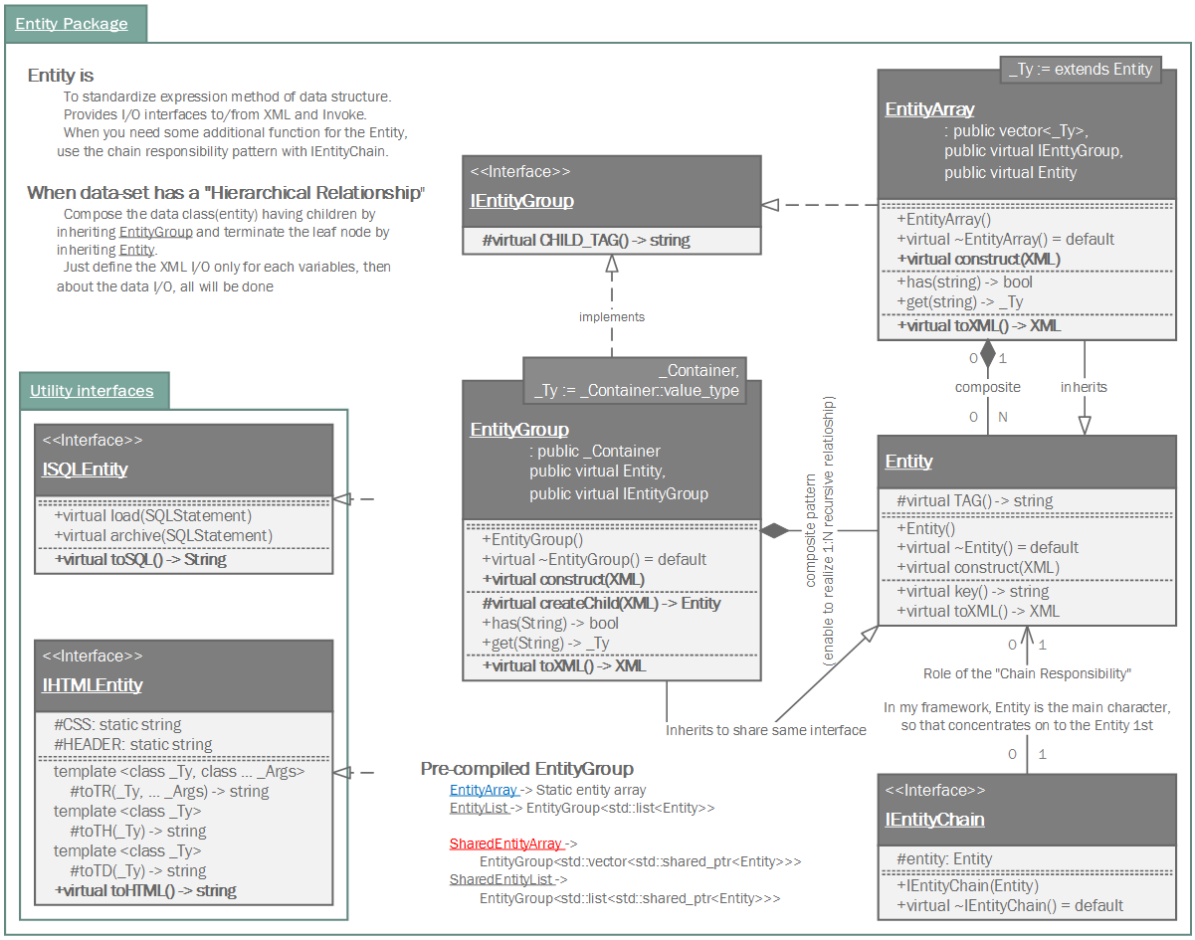
Get a string represents a <tr> tag.

6.66.1 Detailed Description

An interface supporting conversion to html.

IHTMEntity is an interface supporting conversion method to html tags. The IHTMLEntity is used to documentation or printing data of the entity to web-screen; In C++, documentation is main reason to implementing the IHTMLEntity and in JS, printing on web-screen is main reason.

IHTMLEntity provides abstract method toHTML() and varadic template methods for constituting a table. I'm planning to support lots of utility methods for html tags but I've not exprienced developing the homepage or html service. If you're familiar with the html, please contribute to the IHTMLEntity interface.



Example source

```
#include <iostream>

#include <samchon/protocol/Entity.hpp>
#include <samchon/protocol/IHTMLEntity.hpp>
#include <samchon/protocol/SharedEntityArray.hpp>

#include <samchon/library/XML.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
```

```
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;

class Member
    : public Entity, public virtual IHTMLEntity
{
protected:
    typedef Entity super;

    string id;
    string name;
    int age;
    int grade;

public:
    /* ----------------------------------------------------------------------
        CONSTRUCTORS
    ---------------------------------------------------------------------- */
    Member()
        : super(), IHTMLEntity()
    {
    };
    Member(const string &id, const string &name, int age, int grade)
        : super(), IHTMLEntity()
    {
        this->id = id;
        this->name = name;
        this->age = age;
        this->grade = grade;
    };
    virtual ~Member() = default;

    virtual void construct(shared_ptr<XML> xml) override
    {
        this->id = xml->getProperty("id");
        this->name = xml->getProperty("name");
        this->age = xml->getProperty<int>("age");
        this->grade = xml->getProperty<int>("grade");
    };

    /* ----------------------------------------------------------------------
        GETTERSs
    ---------------------------------------------------------------------- */
    virtual auto key() const -> std::string override
    {
        return this->id;
    };

    /* ----------------------------------------------------------------------
        XML EXPORTERS
    ---------------------------------------------------------------------- */
    virtual auto TAG() const -> string override
    {
        return "member";
    };
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("id", id);
        xml->setProperty("name", name);
        xml->setProperty("age", age);
        xml->setProperty("grade", grade);

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        return toTR(id, name, age, grade);
    };
};

class MemberArray
    : public SharedEntityArray<Member>,
    public virtual IHTMLEntity
{
protected:
    typedef SharedEntityArray<Member> super;

    string application;
```

```
    int department;
    Member *chief;

public:
    /* ------------------------------------------------------------------
        CONSTRUCTORS
    ------------------------------------------------------------------ */
    MemberArray()
        : super(), IHTMLEntity()
    {
        this->chief = nullptr;
    };
    virtual ~MemberArray() = default;

    // You don't need to consider children(Member) objects
    // Just concentrate on constructing MemberArray's own member variables
    virtual void construct(shared_ptr<XML> xml) override
    {
        super::construct(xml);

        this->application = xml->getProperty("application");
        this->department = xml->getProperty<int>("department");

        if(xml->hasProperty("chief") == true && this->has( xml->getProperty("chief") ) == true)
            this->chief = this->get( xml->getProperty("cheif") ).get();
    };

protected:
    //FACTORY METHOD FOR MEMBER
    virtual auto createChild(shared_ptr<XML> = nullptr) -> Member* override
    {
        return new Member();
    };

    /* ------------------------------------------------------------------
        XML EXPORTERS
    ------------------------------------------------------------------ */
public:
    virtual auto TAG() const -> string override
    {
        return "memberArray";
    };
    virtual auto CHILD_TAG() const -> string override
    {
        return "member";
    };

    // You don't need to consider children(Member) objects
    // Just concentrate on expressing MemberArray's own member variables
    virtual auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> &xml = super::toXML();
        xml->setProperty("application", application);
        xml->setProperty("department", department);

        if(chief != nullptr)
            xml->setProperty("cheif", chief->key());

        return move(xml);
    };
    virtual auto toHTML() const -> string
    {
        string html = "<table>\n";
        html += toTH("ID", "Name", "Age", "Grade") + "\n";

        for (size_t i = 0; i < 2; i++)
            html += at(i)->toHTML() + "\n";

        html += "</table>";
        return move(html);
    };
};

void main()
{
    string str = string("") +
        "<memberArray application='framework' department='7' cheif='samchon'>\n" +
        "   <member id='samchon' name='Jeongho Nam' age='27' grade='5' />" +
        "   <member id='submaster' name='No Name' age='100' grade='4' />" +
        "   <member id='john' name='John Doe' age='33' grade='2' />" +
        "   <member id='bad_man' name='Bad Man' age='44' grade='-1' />" +
        "   <member id='guest' name='Guest' age='0' grade='0' />" +
        "</memberArray>";
    shared_ptr<XML> xml(new XML(str));

    MemberArray memberArray;
    memberArray.construct(xml);
```

```
    memberArray.emplace_back(new Member("freshman", "A fresh man", 20, 2));
    memberArray.emplace_back(new Member("senior", "A senior", 70, 2));

    cout << memberArray.toXML()->toString() << endl << endl;
    cout << memberArray.toHTML() << endl;
    system("pause");
}
```

Result of the example

```
D:\OneDrive\Project\Samchon\framework\cpp\Debug\entity.exe
<memberArray application="framework" department="7">
        <member age="27" grade="5" id="samchon" name="Jeongho" />
        <member age="100" grade="4" id="submaster" name="No" />
        <member age="33" grade="2" id="john" name="John" />
        <member age="44" grade="-1" id="bad_man" name="Bad" />
        <member age="0" grade="0" id="guest" name="Guest" />
        <member age="20" grade="2" id="freshman" name="A fresh man" />
        <member age="70" grade="2" id="senior" name="A senior" />
</memberArray>

<table>
        <th>
                <td>ID</td>
                <td>Name</td>
                <td>Age</td>
                <td>Grade</td>
        </th>
        <tr>
                <td>samchon</td>
                <td>Jeongho</td>
                <td>27</td>
                <td>5</td>
        </tr>
        <tr>
                <td>submaster</td>
                <td>No</td>
                <td>100</td>
                <td>4</td>
        </tr>
</table>
Press any key to continue . . .
```

Author

Jeongho Nam

6.66.2 Member Function Documentation

6.66.2.1 static auto toTH (const _Ty & *val*, const _Args &... *args*) -> std::string [inline], [static]

Get a string represents a <th> tag.

Template Parameters

_Ty	Type of an argument to be contained with a <td> tag.
_Args	Left varadic template arguments' type

Returns a string represents a title columns with <th> tag by varadic template method.

Supported parameters: that can converted to string

- number
  - (unsigned) short

    - (unsigned) long
    - (unsigned) long long
    - (unsigned) int
    - float
    - double
    - long double
  - string
    - std::string
    - WeakString

Parameters

val	An element to be contained with <td> a tag.
args	Left argument to be contained with <td> tags.

Returns

A string of <th> tag represents title columns.

6.66.2.2 static auto toTR (const _Ty & *val*, const _Args &... *args*) -> std::string [inline], [static]

Get a string represents a <tr> tag.

Template Parameters

_Ty	Type of an argument to be contained with a <tr> tag.
_Args	Left varadic template arguments' type

Returns a string represents a data columns of a row with <tr> tag by varadic template method.

Supported parameters: that can converted to string

  - number
    - (unsigned) short
    - (unsigned) long
    - (unsigned) long long
    - (unsigned) int
    - float
    - double
    - long double
  - string
    - std::string
    - WeakString

Parameters

val	An element to be contained with <tr> a tag.
args	Left argument to be contained with <tr> tags.

Returns

A string of <th> tag represents data columns of a row.

6.66.2.3 virtual auto toHTML ( ) const -> std::string [pure virtual]

Get an html string represents the Entity.

Returns

A string represents the Entity by html tags.

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IHTMLEntity.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IHTMLEntity.cpp

6.67 IndexPair< _Ty > Class Template Reference

A pair of index and its value(_Ty)

```
#include <IndexPair.hpp>
```

Public Member Functions

- auto getIndex () const -> size_t
  Get index.
- auto getValue () -> _Ty &
  Get const value.
- auto getValue () const -> const _Ty &
  Get value.

6.67.1 Detailed Description

template<class _Ty>class samchon::IndexPair< _Ty >

A pair of index and its value(_Ty)

IndexPair is a std::pair<size_t, _Ty> class.

IndexPair expresses index and value. It's useful for expressing finding literals, specified object, calculating mininum or maximum with its position.

[Inherited]

This class couples together a pair of values, which may be of different types (T1 and T2). The individual values can be accessed through its public members first and second.

Pairs are a particular case of tuple.

Referenced comments of std::pair

- http://www.cplusplus.com/reference/utility/pair/

Author

Jeongho Nam

6.67.2 Member Function Documentation

6.67.2.1 auto getIndex ( ) const -> size_t [inline]

Get index.

Returns

Index of the pair

6.67.2.2 auto getValue ( ) -> _Ty& [inline]

Get const value.

Returns

Reference of value of the pair

6.67.2.3 auto getValue ( ) const -> const _Ty& [inline]

Get value.

Returns

Const reference of value of the pair

The documentation for this class was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/IndexPair.hpp

6.68 Instance Class Reference

A physical instance.

`#include <Instance.hpp>`

Public Member Functions

- Instance (const std::string &name, int price, int volume, int weight)

  Construct from instance.
- auto getName () const -> std::string

  Get name.
- auto getPrice () const -> int

  Get price.
- auto getVolume () const -> int

  Get volume.
- auto getWeight () const -> int

  Get weight.
- virtual auto toString () const -> std::string

  Return a string represents the Instance.

Protected Attributes

- std::string name

  Name represent the Instance.
- int price

  Price of an instance -> 1,000 won.
- int volume

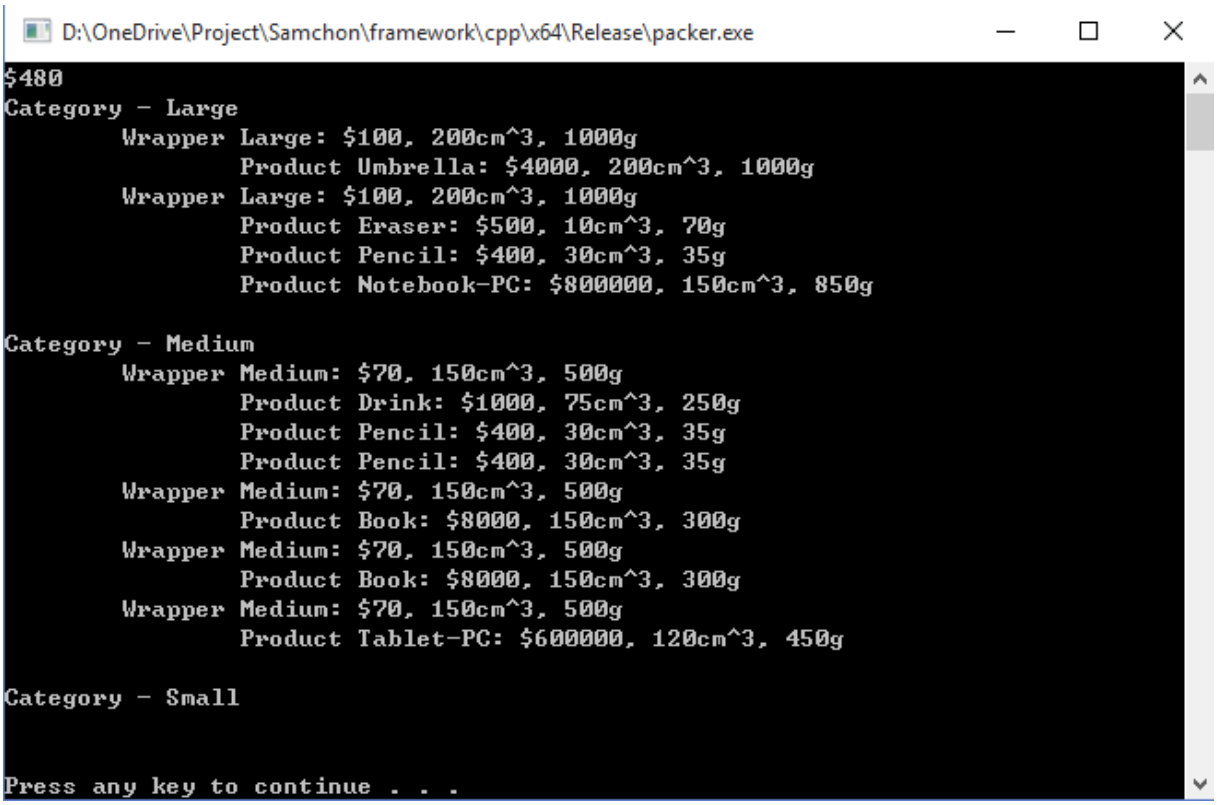
  Volume of an instance -> 130 cm^3.
- int weight

  Weight of an instance -> 1,200 g.

6.68.1 Detailed Description

A physical instance.

A physical instance having its own name, price, volume and weight.



Author

Jeongho Nam

6.68.2 Constructor & Destructor Documentation

6.68.2.1 Instance (const std::string & *name*, int *price*, int *volume*, int *weight*)

Construct from instance.

Parameters

name	Name of the instance
price	Price of the instance
volume	Volume of the instance
weight	Weight of the instance

The documentation for this class was generated from the following files:

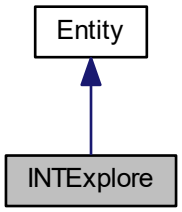
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/Instance.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/Instance.cpp

6.69 INTExplore Class Reference

An interface for exploration.

```
#include <INTExplore.hpp>
```

Collaboration diagram for INTExplore:



Public Member Functions

- INTExplore ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >)

  Construct data of the Entity from an XML object.
- auto getMinimum () const -> double

  Get minimum.
- auto getMaximum () const -> double

  Get maximum.
- auto getSection () const -> unsigned int

  Get section.
- auto getPrecision () const -> int

  Get precision.
- virtual auto toXML () const -> std::shared_ptr< library::XML >

  Get an XML object represents the Entity.

Protected Attributes

- double minimum

  Minimum value.
- double maximum

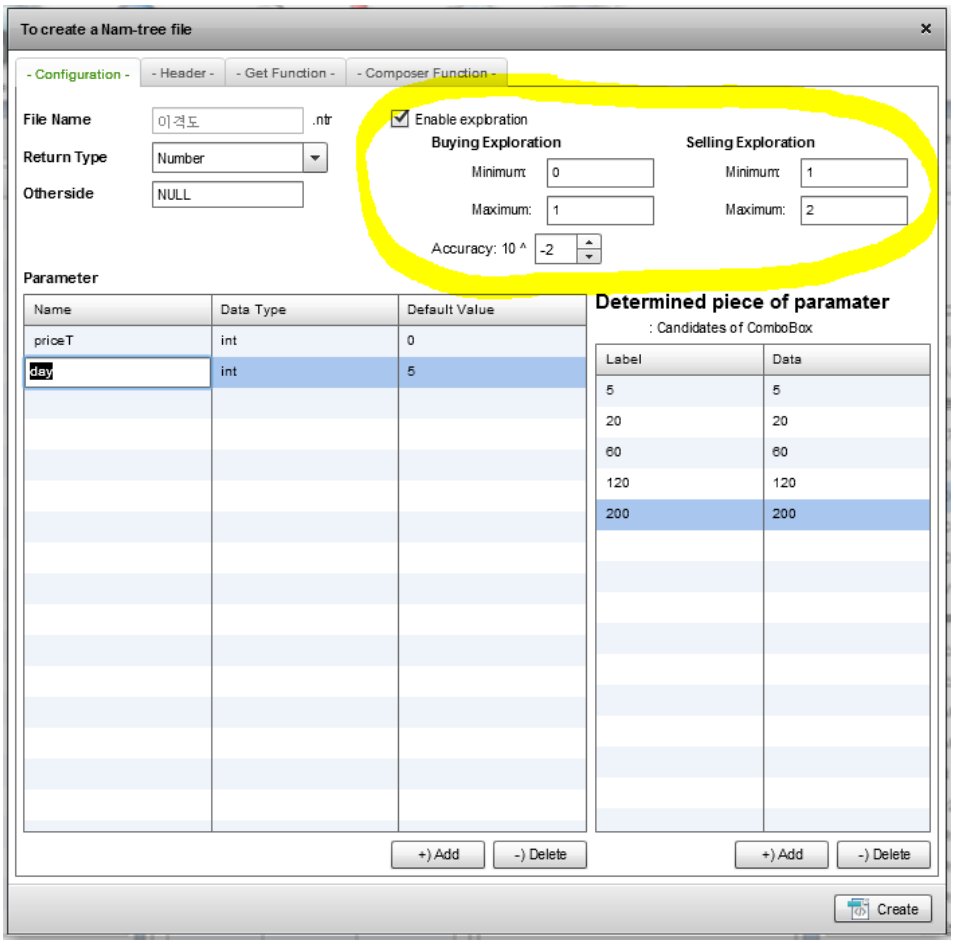
  Maximum value.
- unsigned int section

  Section number of exploration.
- int precision

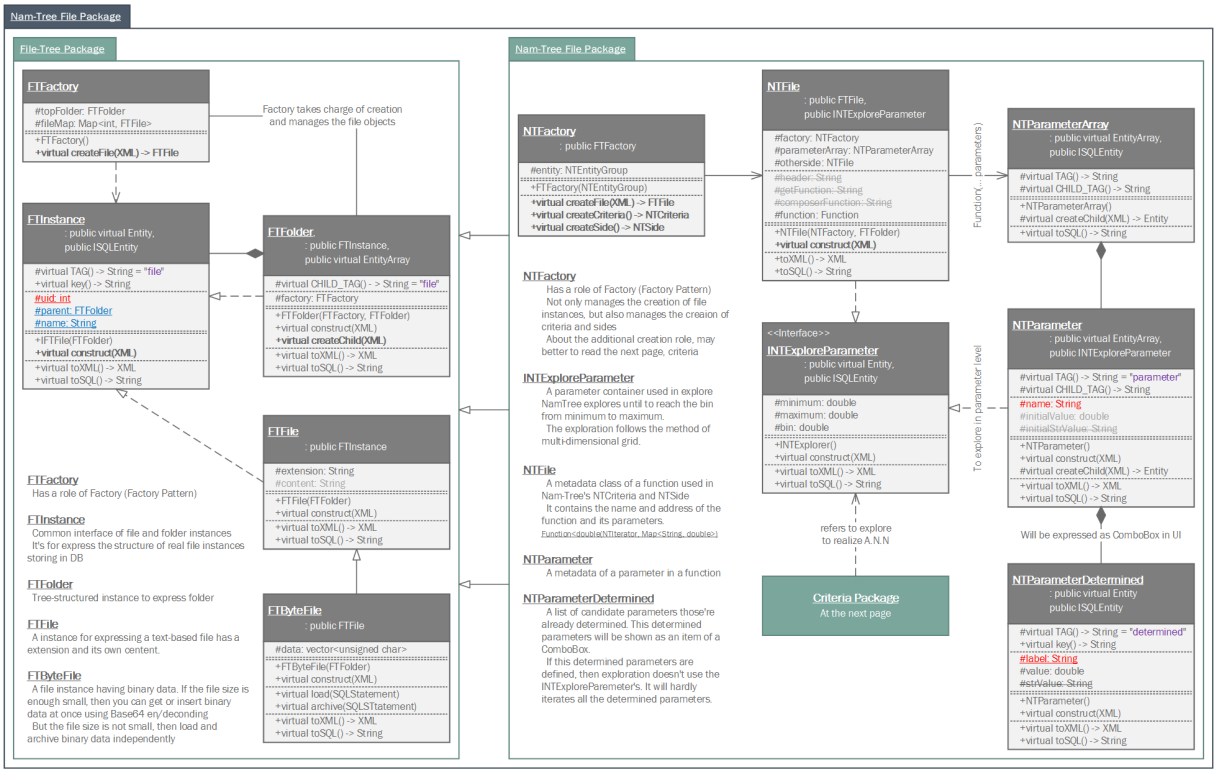
  Target precision: $10^{precision}$

6.69.1 Detailed Description

An interface for exploration.



INTExplore is an interface containing some parameters for grid optimization method.



See also

samchon::namtree

Author

Jeongho Nam

6.69.2 Member Function Documentation

6.69.2.1 void construct (std::shared_ptr< library::XML >) [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

Reimplemented in NTFile, and NTParameter.

6.69.2.2 auto toXML ( ) const -> std::shared_ptr<library::XML> [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

Reimplemented in NTFile, and NTParameter.

The documentation for this class was generated from the following files:

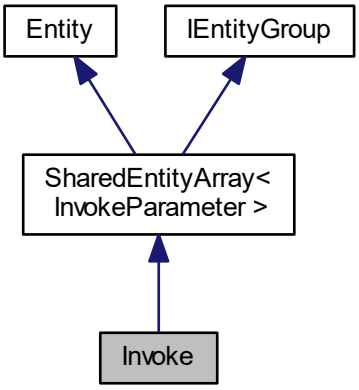
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/INTExplore.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/INTExplore.cpp

6.70 Invoke Class Reference

Standard message of network I/O.

`#include <Invoke.hpp>`

Collaboration diagram for Invoke:



Public Member Functions

- Invoke ()

  Default Constructor.
- Invoke (const std::string &listener)

  Construct from a listener.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- template<typename _Ty , typename... _Args>
  Invoke (const std::string &listener, const _Ty &val, const _Args &...args)

  Construct from arguments.
- auto getListener () const -> std::string

  Get listener.
- void setListener (const std::string &)

  Set listener.
- virtual auto TAG () const -> std::string override

  A tag name when represented by XML.
- virtual auto CHILD_TAG () const -> std::string override

  A tag name of children.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.
- auto toSQL () const -> std::string

  Get a string of sql statement used to archive history log.

Protected Member Functions

- virtual auto createChild (std::shared_ptr< library::XML >) -> InvokeParameter *override

  Factory method of a child Entity.

Protected Attributes

– std::string listener

Represent who listens, often be a function name.

6.70.1 Detailed Description

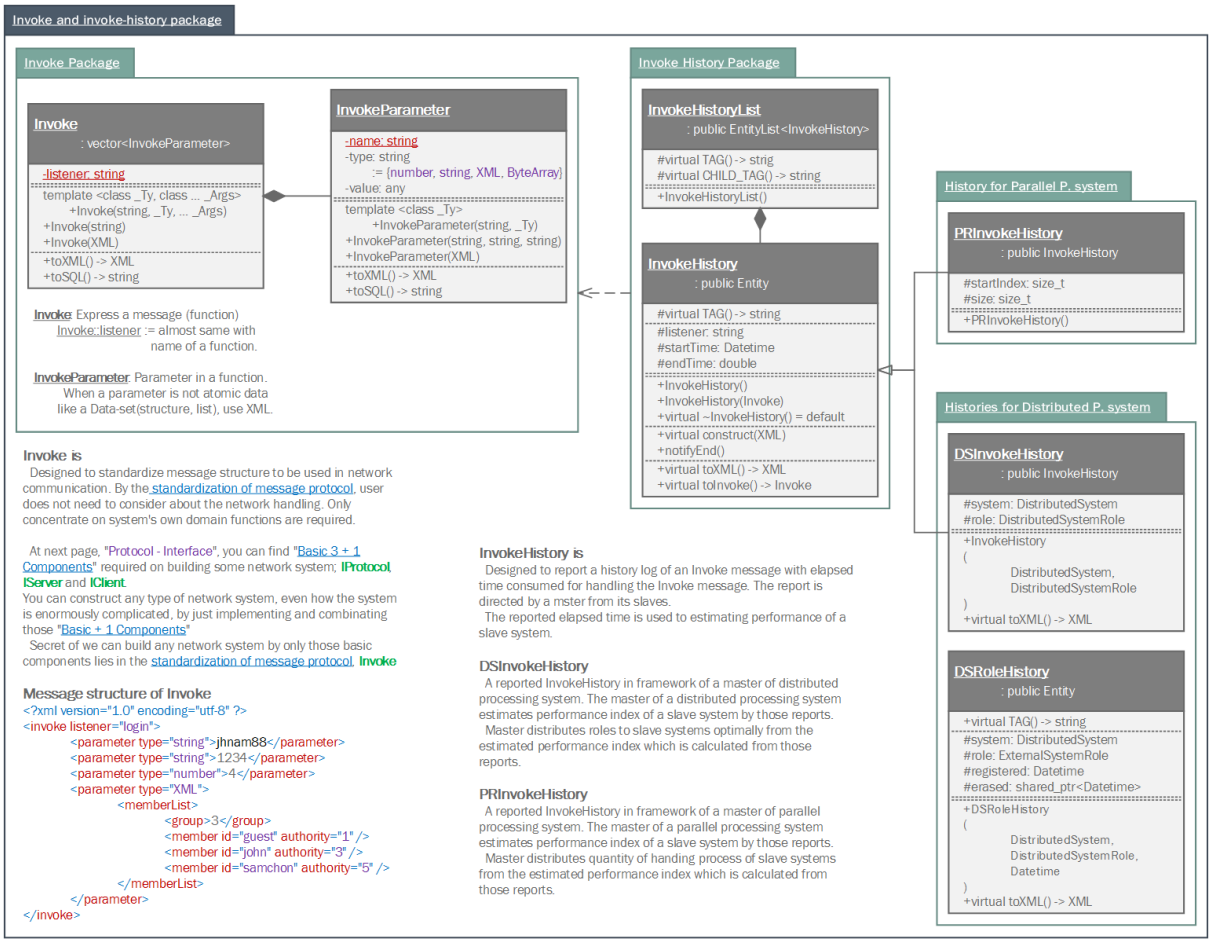
Standard message of network I/O.

Invoke is a class used in network I/O in protocol package of Samchon Framework.

The Invoke message has an XML structure like the result screen of provided example in below. We can enjoy lots of benefits by the normalized and standardized message structure used in network I/O.

The greatest advantage is that we can make any type of network system, even how the system is enourmously complicated. As network communication message is standardized, we only need to concentrate on logical relationships between network systems. We can handle each network system like a object (class) in OOD. And those relationships can be easily designed by using design pattern.

In Samchon Framework, you can make any type of network system with basic 3 + 1 componenets (IProtocol, IServer and IClient + ServerConnector), by implemens or inherits them, like designing classes of S/W architecture.



Example source

```
#include <iostream>

#include <samchon/library/XML.hpp>
#include <samchon/protocol/Invoke.hpp>

using namespace std;
using namespace samchon;
```

```
using namespace samchon::library;
using namespace samchon::protocol;

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

void main()
{
    string str = string("") +
        "<memberList>\n" +
        "   <member id='jhnam88' pass='1231' />\n" +
        "   <member id='samchon' pass='1231'>Administrator</member>\n" +
        "   <group>3</group>\n" +
        "</memberList>";

    shared_ptr<XML> xml(new XML(str));
    shared_ptr<Invoke> invoke(new Invoke("login", "jhnam88", "1231", 4, xml));

    cout << "Invoke to XML: " << endl;
    cout << invoke->toXML()->toString() << endl << endl;

    cout << "------------------------------------------------------------" << endl;
    cout << "   Parameters" << endl;
    cout << "------------------------------------------------------------" << endl;
    cout << "1st param: " << invoke->at(0)->getValue<string>() << endl;
    cout << "2nd param: " << invoke->at(1)->getValue<string>() << endl;
    cout << "3rd param: " << invoke->at(2)->getValue<string>() << endl;
    cout << "4th param: " << endl << invoke->at(3)->getValue<shared_ptr<XML>>()->toString() << endl;

    system("pause");
}
```

Result of the example

```
D:\OneDrive\Project\Samchon\framework\cpp\Debug\invoke.exe
Invoke to XML:
<invoke listener="login">
        <parameter type="string">jhnam88</parameter>
        <parameter type="string">1231</parameter>
        <parameter type="number">4.000000</parameter>
        <parameter type="XML">
                <memberList>
                        <group>3</group>
                        <member id="jhnam88" pass="1231" />
                        <member id="samchon" pass="1231">Administrator</member>
                </memberList>
        </parameter>
</invoke>

------------------------------------------------------------
        Parameters
------------------------------------------------------------
1st param: jhnam88
2nd param: 1231
3rd param: 4
4th param:
<memberList>
        <group>3</group>
        <member id="jhnam88" pass="1231" />
        <member id="samchon" pass="1231">Administrator</member>
</memberList>
Press any key to continue . . .
```

See also

samchon::protocol
protocol::IProtocol

Author

Jeongho Nam

6.70.2 Constructor & Destructor Documentation

6.70.2.1 Invoke (const std::string & *listener*)

Construct from a listener.

Parameters

listener	Represents who listens the Invoke message. Almost same with Function name

6.70.2.2 Invoke (const std::string & *listener,* const _Ty & *val,* const _Args &... *args*) [inline]

Construct from arguments.

Template Parameters

_Ty	Type of an argument which represents a parameter
_Args	Left varadic template arguments' types

Creates Invoke and InvokeParameter(s) at the same time by varadic template method.

By the varadic template constructor, you can't specify name of each InvokeParameter, but specify type and value of each InvokeParameter. If you try to record the Invoke to Database, the name of InvokeParameter will be *NULL*.

Note

By the varadic template constructor, name of InovkeParameter(s) will be omitted. Because of name, an identifier of an InvokeParameter, is omitted, you can't access to InvokeParameter by Invoke::has() or Invoke↩ ::get().

Parameters

listener	A string represents who listens the Invoke message. Almost same with name of a function.
val	A value to be a parameter of Invoke
args	Left arguments to be parameters of Invoke

6.70.3 Member Function Documentation

6.70.3.1 void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

6.70.3.2 auto createChild (std::shared_ptr< library::XML >) -> InvokeParameter *override [override], [protected], [virtual]

Factory method of a child Entity.

EntityGroup::createChild() is a factory method creating a new child Entity which is belonged to the Entity↩ Group. This method is called by EntityGroup::construct(). The children construction methods Entity↩ ::construct() will be called by abstract method of the EntityGroup::construct().

Returns

A new child Entity belongs to EntityGroup.

Implements EntityGroup< _Container, _ETy, _Ty >.

6.70.3.3 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.70.3.4 auto CHILD_TAG ( ) const -> std::string [override], [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

6.70.3.5 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

The documentation for this class was generated from the following files:

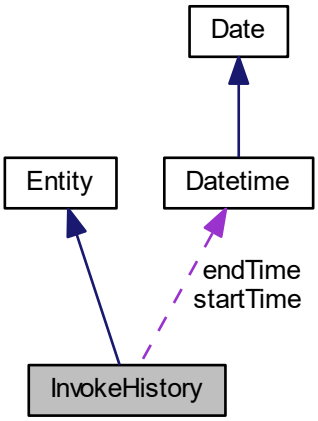
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/Invoke.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/Invoke.cpp

6.71 InvokeHistory Class Reference

A history of an Invoke message.

`#include <InvokeHistory.hpp>`

Collaboration diagram for InvokeHistory:



Public Member Functions

- InvokeHistory ()
  Default Constructor.
- InvokeHistory (std::shared_ptr< Invoke >)
  Construct from an Invoke message.
- virtual void construct (std::shared_ptr< library::XML >) override
  Construct data of the Entity from an XML object.
- void notifyEnd ()
  Notify end of the process.
- virtual auto key () const -> std::string override
  Get a key that can identify the Entity uniquely.
- auto getUID () const -> size_t
  Get uid.
- auto getListener () const -> std::string
  Get listener.
- auto getStartTime () const -> library::Datetime
  Get start time.
- auto getEndTime () const -> library::Datetime
  Get end time.
- auto calcElapsedTime () const -> long long
  Get elapsed time.
- virtual auto TAG () const -> std::string override
  A tag name when represented by XML.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the Entity.
- auto toInvoke () const -> std::shared_ptr< Invoke >
  Get an Invoke message.

Protected Attributes

- size_t uid

  An identifier.
- std::string listener

  A listener of the Invoke message.
- library::Datetime startTime

  Start time of the history.
- library::Datetime endTime

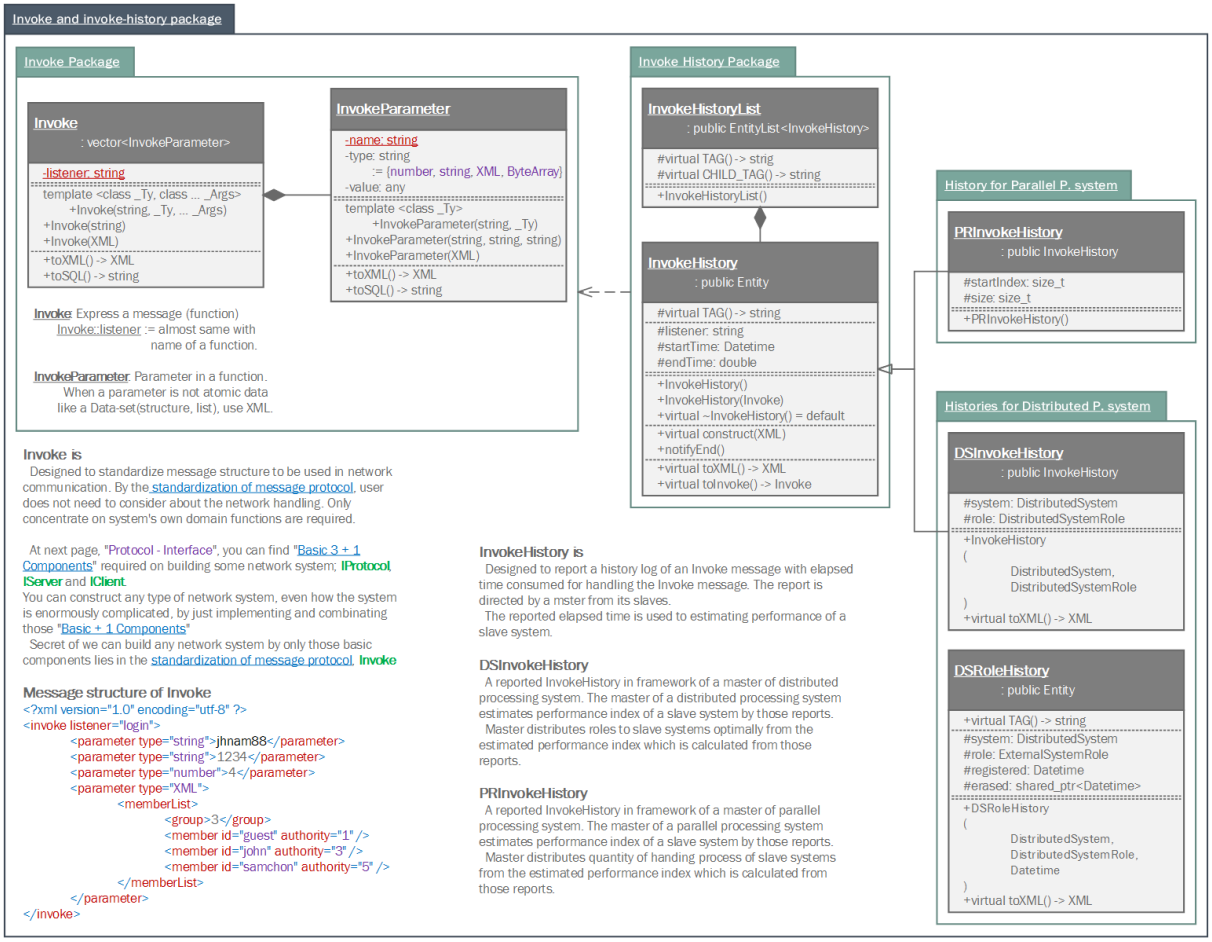
  End time of the history.

6.71.1 Detailed Description

A history of an Invoke message.

InvokeHistory is a class for reporting history log of an Invoke message with elapsed time from a slave to its master.

With the elapsed time, consumed time for a process of handling the Invoke message, InvokeHistory is reported to the master. The master utilizies the elapsed time to estimating performances of each slave system. With the estimated performan index, master retrives the optimal solution of distributing processes.



See also

samchon::protocol
samchon::protocol::master
samchon::protocol::slave

Author

Jeongho Nam

6.71.2 Constructor & Destructor Documentation

6.71.2.1 InvokeHistory (std::shared_ptr< Invoke >)

Construct from an Invoke message.

InvokeHistory does not archive entire Invoke message, only archives its listener.

Parameters

invoke	A message to archive its history log

6.71.3 Member Function Documentation

6.71.3.1 void construct (std::shared_ptr< library::XML >) [override], [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

Reimplemented in PRInvokeHistory.

6.71.3.2 void notifyEnd ( )

Notify end of the process.

Notifies end of a process handling the matched Invoke message to InvokeHistory.

InvokeHistory archives the end datetime and calculates elapsed time as nanoseconds.

6.71.3.3 auto key ( ) const -> std::string [override], [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented from Entity.

6.71.3.4 auto calcElapsedTime ( ) const -> long long

Get elapsed time.

Calculates elapsed time frrom start, end time and Returns it.

The elapsed time will be reported to a master and the master will utilize the elapsed time to estimating performance of the slave system, the InvokeHistory is belonged to.

Returns

An elapsed time expressed as nano seconds.

6.71.3.5 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.71.3.6 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

Reimplemented in PRInvokeHistory.

6.71.3.7 auto toInvoke ( ) const -> std::shared_ptr<Invoke>

Get an Invoke message.

Returns an Invoke message to report to a master that how much time was elapsed on a process handling the Invoke message. In master, those reports are used to estimate performance of each slave system.

Returns

An Invoke message to report master.

6.71.4 Member Data Documentation

6.71.4.1 std::string listener [protected]

A listener of the Invoke message.

InvokeHistory does not archive entire data of an Invoke message. InvokeHistory only archives its listener. The first, formal reason is to save space, avoid wasting spaces.

The second, complicate reason is on an aspect of which systems are using the InvokeHistory class. InvokeHistory is designed to let slave reports to master elapsed time of a process used to handling the Invoke message. If you want to archive entire history log of Invoke messages, then the subject should be master, not the slave using InvokeHistory classes.

6.71.4.2 library::Datetime startTime [protected]

Start time of the history.

Means start time of a process handling the Invoke message. The start time not only has ordinary arguments represented Datetime (year to seconds), but also has very precise values under seconds, which is expressed as nano seconds (10^-9).

The precise start time will be used to calculate elapsed time with end time.

6.71.4.3 library::Datetime endTime [protected]

End time of the history.

Means end time of a process handling the Invoke message. The end time not only has ordinary arguments represented Datetime (year to seconds), but also has very precise values under seconds, which is expressed as nano seconds (10^-9).

The precise end time will be used to calculate elapsed time with start time.

The documentation for this class was generated from the following files:

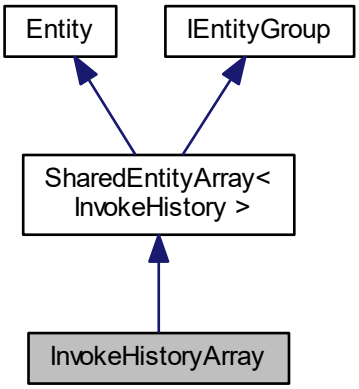
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/InvokeHistory.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/InvokeHistory.cpp

6.72 InvokeHistoryArray Class Reference

An array of InvokeHistory.

`#include <InvokeHistoryArray.hpp>`

Collaboration diagram for InvokeHistoryArray:



Public Member Functions

- InvokeHistoryArray ()

  Default Constructor.
- virtual auto TAG () const -> std::string

  A tag name when represented by XML.
- virtual auto CHILD_TAG () const -> std::string

  A tag name of children.

Protected Member Functions

- virtual auto createChild (std::shared_ptr< library::XML >) -> InvokeHistory *override
  Factory method of a child Entity.

6.72.1 Detailed Description

An array of InvokeHistory.

Author

Jeongho Nam

6.72.2 Member Function Documentation

6.72.2.1 auto createChild (std::shared_ptr< library::XML >) -> InvokeHistory *override [override], [protected], [virtual]

Factory method of a child Entity.

EntityGroup::createChild() is a factory method creating a new child Entity which is belonged to the EntityGroup. This method is called by EntityGroup::construct(). The children construction methods Entity::construct() will be called by abstract method of the EntityGroup::construct().

Returns

A new child Entity belongs to EntityGroup.

Implements EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in PRInvokeHistoryArray.

6.72.2.2 auto TAG ( ) const -> std::string [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.72.2.3 auto CHILD_TAG ( ) const -> std::string [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

The documentation for this class was generated from the following files:

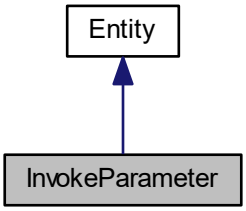
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/InvokeHistoryArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/InvokeHistoryArray.cpp

6.73 InvokeParameter Class Reference

A parameter of an Invoke.

`#include <InvokeParameter.hpp>`

Collaboration diagram for InvokeParameter:



Public Member Functions

- InvokeParameter (const std::string &, const std::string &, const std::string &)
  Construct from arguments.
- template<typename _Ty >
  InvokeParameter (const std::string &name, const _Ty &val)
  Construct with its name and a value.
- InvokeParameter (const std::string &, std::string &&)
  Construct from the name and a moved string.
- InvokeParameter (const std::string &, ByteArray &&)
  Construct from name and a moved ByteArray.
- virtual auto key () const -> std::string override
  Get a key that can identify the Entity uniquely.
- auto getName () const -> std::string
  Get name.
- auto getType () const -> std::string
  Get type.
- template<typename _Ty >
  auto getValue () const -> _Ty
  Get value.
- auto getvalueAsXML () const -> std::shared_ptr< library::XML >
  Get value as XML object.
- template<typename _Ty >
  auto referValue () const -> const _Ty &
  Reference value.
- template<typename _Ty >
  auto moveValue () -> _Ty
  Move value.

Protected Member Functions

- InvokeParameter ()
  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override
  Construct data of the Entity from an XML object.

- virtual auto TAG () const -> std::string override

  A tag name when represented by XML.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the Entity.
- auto toSQL () const -> std::string

  Get a string of sql statement used to archive history log.

Protected Attributes

- std::string name

  A name can represent the parameter.
- std::string type

  Type of the parameter.
- std::string str

  A string value if the type is "string" or "number".
- std::shared_ptr< library::XML > xml

  An XML object if the type is "XML".
- std::unique_ptr< ByteArray > byteArray

  A binary value if the type is "ByteArray".

6.73.1 Detailed Description

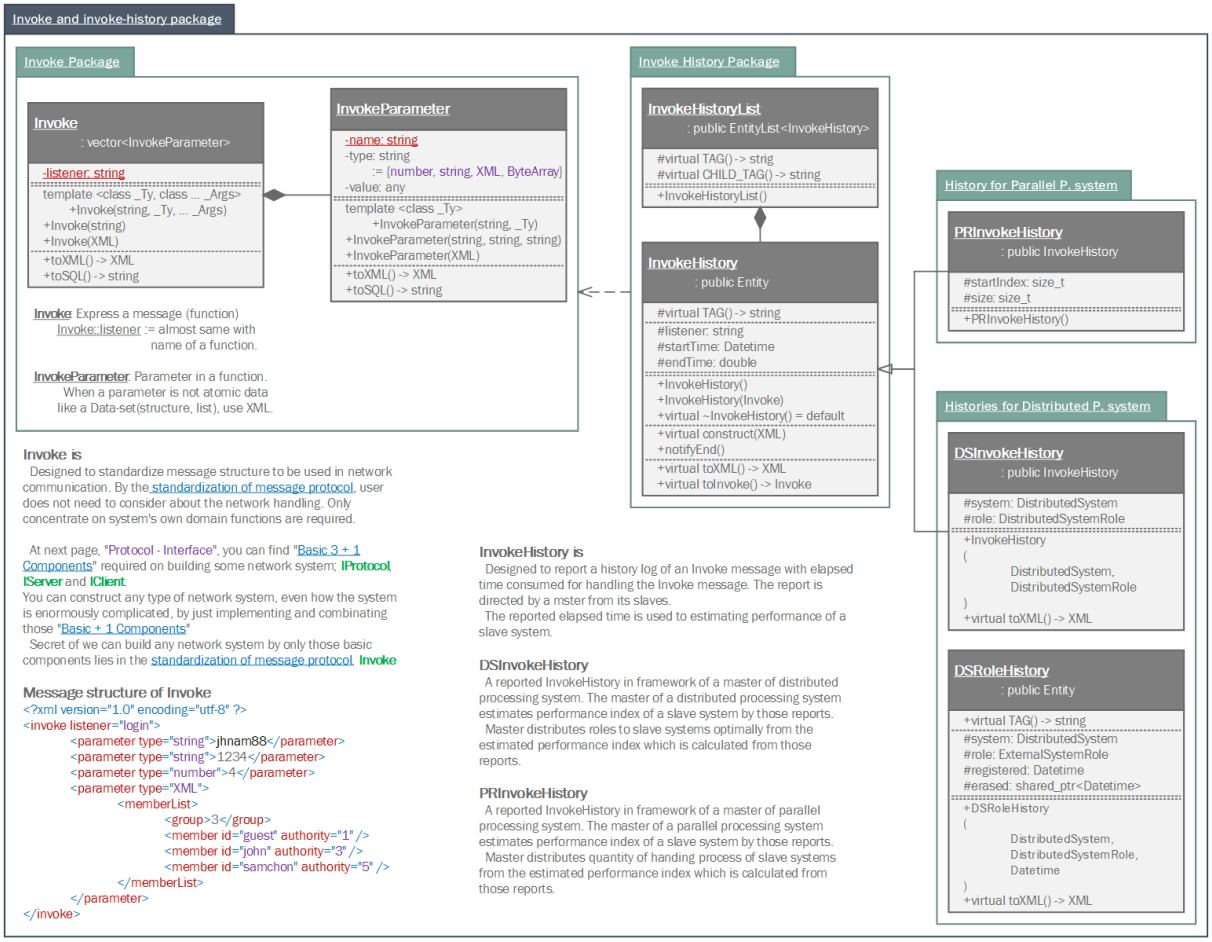
A parameter of an Invoke.

A parameter with its name, type and value in an Invoke message.

InvokeParameter supports 4 types pre-defined.

- number
- string
- XML
- ByteArray

You can specify InvokeParameter to have some type which is not one of the basic 4 types, but values of the custom type must be enable to expressed by one of those types; number, string, XML and ByteArray.



Note

A member variable void∗ representing any type of value is depreciated.

Since C++11, method of void pointer (Declare a variable as void pointer whose type can not specified. And convert the pointer type to specified one for each case) is recommended to avoid.

As that reason, the *void value was a candidate to be depreciated or to be replaced to boost::any. And the void value* is really depreciated since version v1.0.

See also

protocol::Invoke
samchon::protocol

Author

Jeongho Nam

6.73.2 Constructor & Destructor Documentation

6.73.2.1 InvokeParameter (const std::string & , const std::string & , const std::string &)

Construct from arguments.

Specifies all arguments of an InvokeParameter. You can specify a custom type, that is not one of number or string, but value of the type must be enable to expressed by a string.

If you want to express an object or container group, use XML instead; *Invoke::InvokeParameter(string, XML)*.

Parameters

name	A name can represent the InvokeParameter
type	Type of value in the InvokeParameter
value	A value capsuled by a string

6.73.2.2 InvokeParameter (const std::string & *name*, const _Ty & *val*) [inline]

Construct with its name and a value.

Type of the InvokeParameter will be determined automatically.

Type supported in the template constructor:

- number
  - (unsigned) short
  - (unsigned) long
  - (unsigned) long long
  - (unsigned) int
  - float
  - double
  - long double
- string
  - std::string
  - WeakString
- XML
  - std::shared_ptr<library::XML>
  - Entity; protocol::Entity::toXML()
- ByteArray

Template Parameters

_Ty	Type of value

6.73.3 Member Function Documentation

6.73.3.1 void construct (std::shared_ptr< library::XML >) [override], [protected], [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

6.73.3.2 auto key ( ) const -> std::string [override], [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented from Entity.

6.73.3.3 auto getValue ( ) const -> _Ty [inline]

Get value.

Template Parameters

_Ty	Type of value to get

6.73.3.4 auto getvalueAsXML ( ) const -> std::shared_ptr<library::XML>

Get value as XML object.

Same with getValue< std::shared_ptr<library::XML> >();

6.73.3.5 auto referValue ( ) const -> const _Ty &

Reference value.

Template Parameters

_Ty	Type of value to reference

6.73.3.6 auto moveValue ( ) -> _Ty

Move value.

Template Parameters

_Ty	Type of value to move

6.73.3.7 auto TAG ( ) const -> std::string [override], [protected], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.73.3.8 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [protected], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

6.73.4 Member Data Documentation

6.73.4.1 std::string type [protected]

Type of the parameter.

List of pre-defined types

- number
- string
- XML
- ByteArray

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/InvokeParameter.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/InvokeParameter.cpp

6.74 IOperator< _Ty > Class Template Reference

Interface for comparision operator.

`#include <IOperator.hpp>`

Public Member Functions

- IOperator ()
  Default Constructor.
- virtual auto operator< (const IOperator &) const -> bool=0
  Less.
- virtual auto operator== (const IOperator &) const -> bool=0
  Equal.

6.74.1 Detailed Description

template<typename _Ty>class samchon::library::IOperator< _Ty >

Interface for comparision operator.

IOperator is a template class makes enalbe to realizing all compare operators by only overriding operator< and operator== methods.

- [!=] -> [NOT ==]
- [<=] -> [< OR ==]
- [>] -> [NOT < AND NOT ==]
- [>=] -> [NOT <]

Methods operator< and operator== are abstract. Overrides them.

Note

If you want to realize all comparison operators without inheritance and specifying clear type of parameter, you can use following macros instead.

- OPERATOR_METHODS_INLINE(_Ty)
- OPERATOR_METHODS_HEADER(_Ty)
- OPERATOR_METHODS_BODY(_Class, _Ty)

See also

samchon::library

Author

Jeongho Nam

6.74.2 Member Function Documentation

6.74.2.1 virtual auto operator< (const IOperator< _Ty > &) const -> bool [pure virtual]

Less.

Overrides method less.

Parameters

obj	The object to compare.

Returns

The object is less than this.

6.74.2.2 virtual auto operator== (const IOperator< _Ty > &) const -> bool [pure virtual]

Equal.

Overrides methods equal.

Parameters

obj	The object to compare.

Returns

The object is equal with this.

The documentation for this class was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/IOperator.hpp

6.75 IProtocol Class Reference

An interface of Invoke message chain.

```
#include <IProtocol.hpp>
```

Public Member Functions

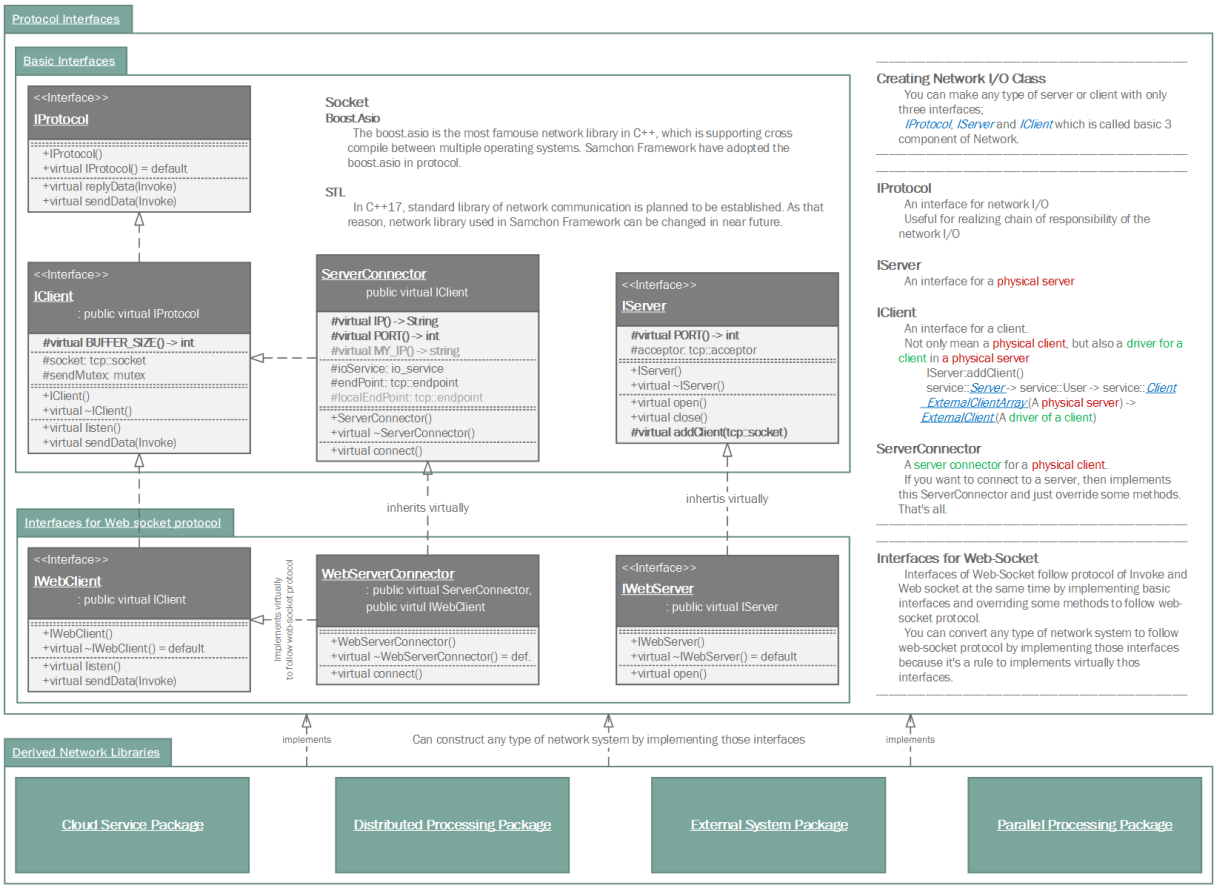
- IProtocol ()
  Default Constructor.
- virtual void replyData (std::shared_ptr< Invoke >)
  Reply a message.
- virtual void sendData (std::shared_ptr< Invoke >)
  Send a message.

6.75.1 Detailed Description

An interface of Invoke message chain.

IProtocol is an interface for Invoke message, which is standard message of network I/O in Samchon Framework, chain. The IProtocol interface is used to network drivers and some classes which are in a relationship of chain of responsibility with those network drivers.

IProtocol is one of the basic 3 + 1 components that can make any type of network system in Samchon Framework with IServer and IClient. Following the "chain of responsibility" pattern, looking around classes in Samchon Framework, you can see all related classes with network I/O are implemented from the IProtocol.



Author

Jeongho Nam

6.75.2 Member Function Documentation

6.75.2.1 virtual void replyData (std::shared_ptr< Invoke >) [virtual]

Reply a message.

Handles a replied Invoke message or shifts the responsibility to related chain.

Parameters

invoke	An Invoke message gotten from a network system

Reimplemented in User, Client, DistributedSystem, ExternalSystemArray, ExternalSystem, DistributedSlaveSystemMediator, and ParallelSlaveSystemMediator.

6.75.2.2 virtual void sendData (std::shared_ptr< Invoke >) [virtual]

Send a message.

Sends Invoke message to a network system or shifts the responsibility to related chain.

Parameters

invoke	An Invoke message to send to a network system

Reimplemented in User, DistributedSystem, Client, ExternalSystemRole, ExternalSystemArray, Distributed↩ SystemRole, Service, IClient, and IWebClient.

The documentation for this class was generated from the following files:

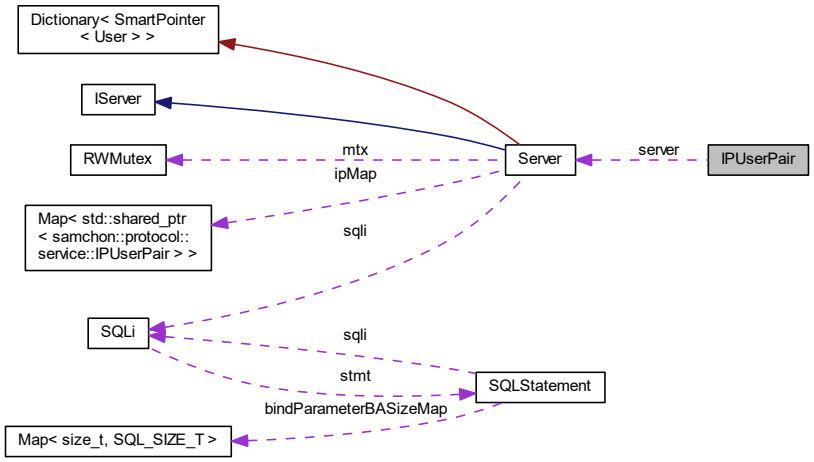
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IProtocol.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IProtocol.cpp

6.76 IPUserPair Class Reference

Relationship between IP address and User for issuing session.

`#include <IPUserPair.hpp>`

Collaboration diagram for IPUserPair:



Private Member Functions

- IPUserPair (Server *server, const std::string &)

  Construct from Server and IP address.
- auto getSessionID (Socket *, size_t) -> std::string

  Get a session ID.
- auto issueSessionID (size_t) const -> std::string

  Issue a session ID.

Private Attributes

- Server * server

  The Server containing IP addresses and User(s)
- std::string ip

An IP address.

– std::set< const User * > userSet

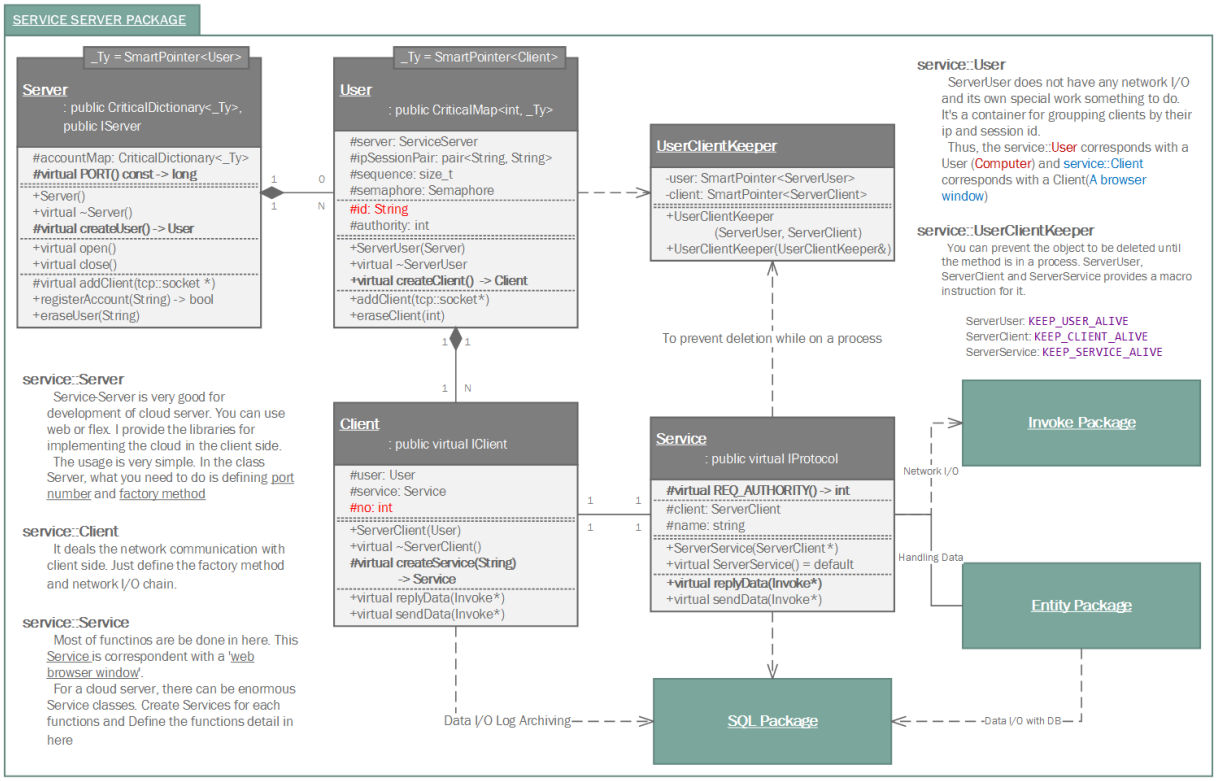
User(s) belongs to the IP address.

6.76.1 Detailed Description

Relationship between IP address and User for issuing session.

IPUserPair is a class representing 1:N relationship between IP address and User.

The key role of IPUserPair is to issuing session ID, that is published to User(s), within a critical section of an IP address.



Note

Samchon Framework is for Cloud Server basically, so I've adopted Session ID to discriminate unique User to follow standard. But I think there can be another good strategies something like a using MAC address instead of publishing Session-ID to discriminate uniqueness of User. It's okay to adapt those strategies and rebuild my framework's source to be suitable for those.

Warning

When changing the strategy, you've to careful about policy of each system or libraries.

Followings are something to consider when adjusting MAC address for uniqueness.

– In server side and view of socket, boost::asio or candidate network i/o libary of stl does not allow accessing otherside system's MAC address. But basic socket of window or linux are allowed.

– In client side and view of browser, IE allows to get MAC address by javascript with ActiveX, but other browsers do not allow.

See also

samchon::protocol
samchon::protocol::service

Author

Jeongho Nam

6.76.2 Member Function Documentation

6.76.2.1 auto getSessionID (Socket * *socket,* size_t *sequence*) -> std::string [private]

Get a session ID.

Asks Whether client to have a session id stored in cookie.

If client has, get the session id from client, but if not, issues a new session ID and notify it(enrolled new session ID) to client. Additionally, if the session id was gotten from a client, inspect validate of the session ID.

Those tasks will hold a critical section of a global IP address

Warning

The session id must be unique

Returns

A new session ID that can ensure uniqueness

6.76.2.2 auto issueSessionID (size_t *sequence*) const -> std::string [private]

Issue a session ID.

Issues a session ID when a target client haven't issued seesion ID yet

Default organization of the Session ID

- {$name}::{$port}::{$sequence}::{$linux_time}
  1. $name: Name of the Server; Server::NAME()
  2. $port: Port number of the Server; Server::PORT()
  3. $sequence: Sequence number of creation of User in Server
  4. $linux_time: Linux time of current time.
     If the object Server is re-constructed, $sequence can be duplicated. $linux_time is to avoid the duplication so that can ensure uniqueness on session id

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/IPUserPair.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/IPUserPair.cpp

6.77 IServer Class Reference

An interface of a physical server.

```
#include <IServer.hpp>
```

Public Member Functions

- IServer ()

  Default Constructor.
- virtual void open ()

  Open the server.
- virtual void close ()

  Close the server.

Protected Member Functions

- virtual auto MY_IP () const -> std::string

  (optional) Server's IP
- virtual auto PORT () const -> int=NULL

  Port number of the server.
- virtual void addClient (Socket *)=0

  Handling connection of a physical client.

Protected Attributes

- Acceptor * acceptor

  An acceptor for clients.

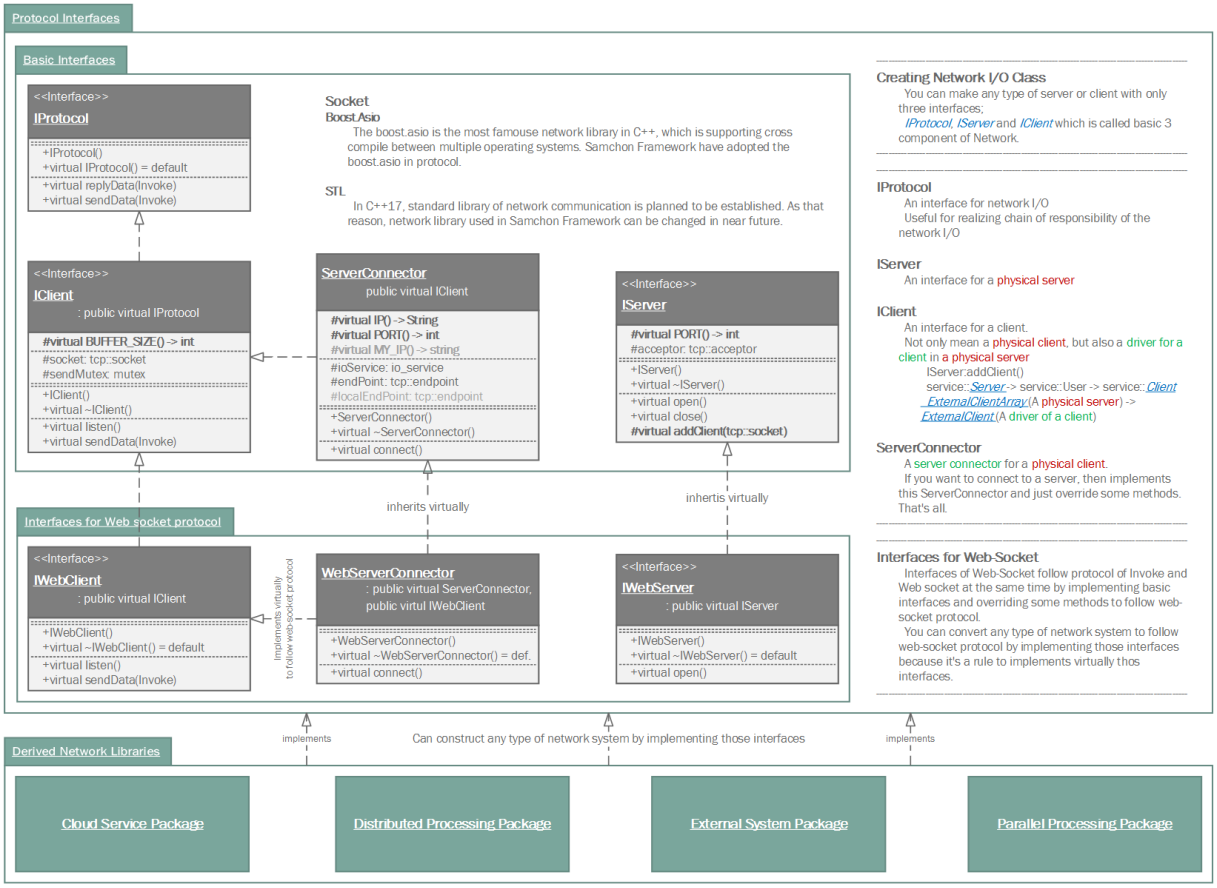
6.77.1 Detailed Description

An interface of a physical server.

IServer provides methods for opening a server.

IServer is one of the basic 3 + 1 components that can make any type of network system in Samchon Framework with IProtocol and IClient. Looking around classes in Samchon Framework, you can see all servers are implemented from the IServer.

When a client connects to the server, the abstract method IServer::addClient() is called with a new thread. If you want to accept only a client at a time, use OneToOneServer instead.



Example source

A simple chat server running on console

examples/console_chat_server/ChatServer.hpp

```
1 #pragma once
2 #include <samchon/protocol/IServer.hpp>
3 #include <samchon/protocol/IProtocol.hpp>
4
5 #include <set>
6 #include <samchon/library/RWMutex.hpp>
7
8 namespace samchon
9 {
10     namespace example
11     {
21         namespace console_chat_server
22         {
23             class ChatClient;
24
30             class ChatServer
31                 : public protocol::IServer,
32                 public protocol::IProtocol
33             {
34             private:
35                 typedef protocol::IServer super;
36
40                 std::set<ChatClient*> clientSet;
41
45                 library::RWMutex rwMutex;
46
47             public:
51                 ChatServer();
52                 virtual ~ChatServer() = default;
53
57                 void eraseClient(ChatClient *client);
58
59             protected:
63                 virtual auto PORT() const -> int override
64                 {
65                     return 37749;
66                 };
```

```
67
71                  virtual void addClient(protocol::Socket *socket) override;
72
73              public:
80                  virtual void replyData(std::shared_ptr<protocol::Invoke>) override;
81
88                  virtual void sendData(std::shared_ptr<protocol::Invoke>) override;
89              };
90          };
91      };
92 };
```

examples/console_chat_server/ChatServer.cpp

```
1 #include <samchon/examples/console_chat_server/ChatServer.hpp>
2 #include <samchon/examples/console_chat_server/ChatClient.hpp>
3
4 #include <thread>
5 #include <samchon/protocol/Invoke.hpp>
6
7 using namespace std;
8 using namespace samchon::library;
9 using namespace samchon::protocol;
10 using namespace samchon::example::console_chat_server;
11
12 ChatServer::ChatServer()
13     : super()
14 {
15 }
16 void ChatServer::addClient(Socket *socket)
17 {
18     ChatClient *client = new ChatClient(this, socket);
19
20     UniqueWriteLock uk(rwMutex);
21     clientSet.insert(client);
22
23     thread(&ChatClient::listen, client).detach();
24 }
25 void ChatServer::eraseClient(ChatClient *client)
26 {
27     UniqueWriteLock uk(rwMutex);
28     clientSet.erase(client);
29 }
30
31 void ChatServer::replyData(shared_ptr<Invoke> invoke)
32 {
33     sendData(invoke);
34 }
35 void ChatServer::sendData(shared_ptr<Invoke> invoke)
36 {
37     UniqueReadLock uk(rwMutex);
38
39     for(auto it = clientSet.begin(); it != clientSet.end(); it++)
40         (*it)->sendData(invoke);
41 }
```

examples/console_chat_server/ChatClient.hpp

```
1 #pragma once
2 #include <samchon/protocol/IClient.hpp>
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace console_chat_server
9         {
10             class ChatServer;
11
21             class ChatClient
22                 : public protocol::IClient
23             {
24             private:
25                 typedef protocol::IClient super;
26
30                 ChatServer *server;
31
32             public:
39                 ChatClient(ChatServer*, protocol::Socket*);
40
45                 virtual ~ChatClient();
46
51                 virtual void replyData(std::shared_ptr<protocol::Invoke>) override;
52             };
53         };
54     };
```

```
};
```

examples/console_chat_server/ChatClient.cpp

```
#include <samchon/examples/console_chat_server/ChatClient.hpp>
#include <samchon/examples/console_chat_server/ChatServer.hpp>

#include <samchon/protocol/Invoke.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;
using namespace samchon::example::console_chat_server;

ChatClient::ChatClient(ChatServer *server, Socket *socket)
    : super()
{
    this->server = server;
    this->socket = socket;
}
ChatClient::~ChatClient()
{
    server->eraseClient(this);
}

void ChatClient::replyData(shared_ptr<Invoke> invoke)
{
    server->replyData(invoke);
}
```

examples/console_chat_server/main.cpp

```
#include <iostream>
#include <samchon/examples/console_chat_server/ChatServer.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace samchon::example::console_chat_server;

void main()
{
    ChatServer chatServer;
    chatServer.open();

    system("pause");
}
```

Author

Jeongho Nam

6.77.2 Member Function Documentation

6.77.2.1 void open ( ) [virtual]

Open the server.

Note

It holds (monopolies) a thread.

Reimplemented in IWebServer.

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IServer.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IServer.cpp

6.78 ISQLEntity Class Reference

An interface supporting DB-I/O.

```
#include <ISQLEntity.hpp>
```

Public Member Functions

- ISQLEntity ()

  Default Constructor.
- virtual void load (std::shared_ptr< library::SQLStatement > stmt)

  Load data of entity from DB.
- virtual void archive (std::shared_ptr< library::SQLStatement > stmt)

  Archive data of entity to DB.
- virtual auto toSQL () const -> std::string

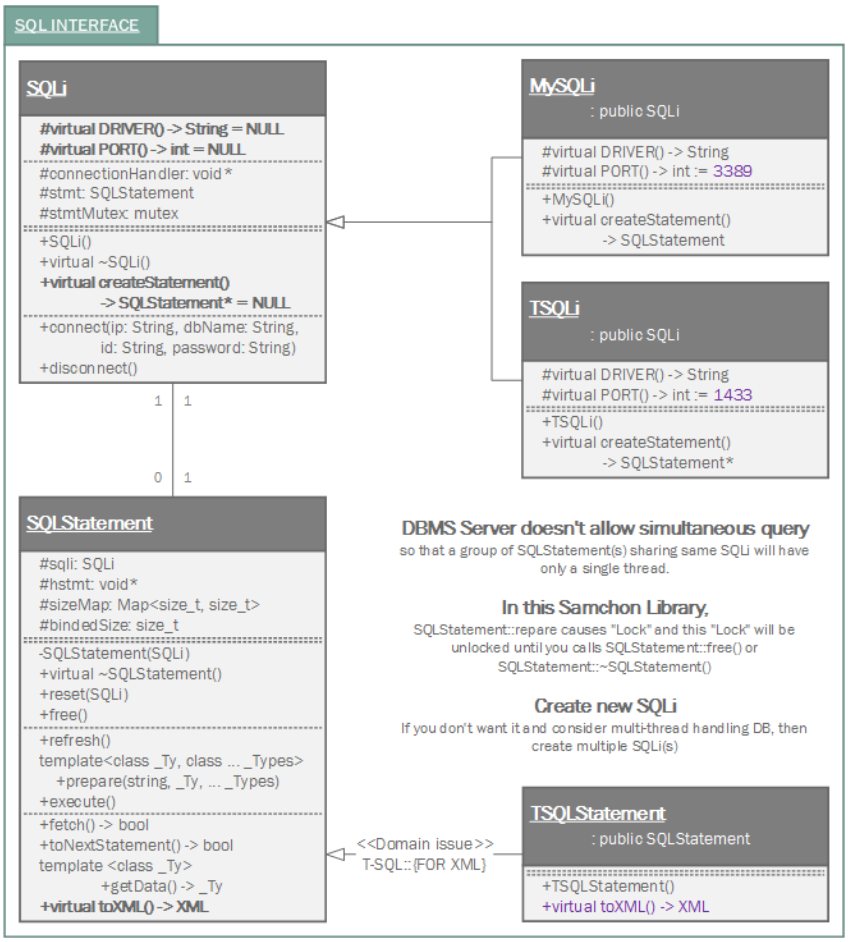
  Get a sql-statement string represents the entity.

6.78.1 Detailed Description

An interface supporting DB-I/O.

ISQLEntity is an interface for interacting with Database. By implementing load() and archive() methods, you can make an entity to interact with a Database.

If an entity implementing the ISQLEntity is a type of EntityGroup and its children also implemented the IS↩QLEntity, you can realize construction and archiving chain by each child entity to do their I/O procedures by their own.



Author

Jeongho Nam

6.78.2 Member Function Documentation

6.78.2.1 void load (std::shared_ptr< library::SQLStatement > *stmt*) [virtual]

Load data of entity from DB.

Loads and constructs data of the entity from a SQLStatement.

If an entity implementing the ISQLEntity is a type of EntityGroup and its children also implemented the ISQLEntity, you can realize construction chain by each child entity fetches data of their own by themselves.

Parameters

stmt	SQLStatement storing data of the Entity

6.78.2.2 void archive (std::shared_ptr< library::SQLStatement > *stmt*) [virtual]

Archive data of entity to DB.

Archives data of the entity to Database by a SQLStatement.

If an entity implementing the ISQLEntity is a type of EntityGroup and its children also implemented the ISQLEntity, you can realize archiving chain by each child entity archives data of their own by themselves.

Parameters

stmt	SQLStatement would store data of the Entity

6.78.2.3 auto toSQL ( ) const -> std::string [virtual]

Get a sql-statement string represents the entity.

ISQLEntity::toSL() is an abstract method returns a sql-statement string.

However, if there's some parameter in a sql-statement string from the toString() method, it is categorized in dynamic sql. In that case, do not access table directly but indirect from sql-procedure. Indirecting procedure is good for performance and security.

Returns

SQL storing record(s) to temporary table of a Procedure

The documentation for this class was generated from the following files:

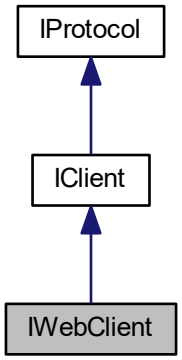
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ISQLEntity.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ISQLEntity.cpp

6.79 IWebClient Class Reference

An interface for a web-client.

```
#include <IWebClient.hpp>
```

Collaboration diagram for IWebClient:



Public Member Functions

- IWebClient ()

  Default Constructor.
- virtual void listen () override

  Listens message from a related system.
- virtual void sendData (std::shared_ptr< Invoke >) override

  Send Invoke message following web-socket protocol.

Additional Inherited Members

6.79.1 Detailed Description

An interface for a web-client.

IWebClient is a IClient following web-socket protocol.

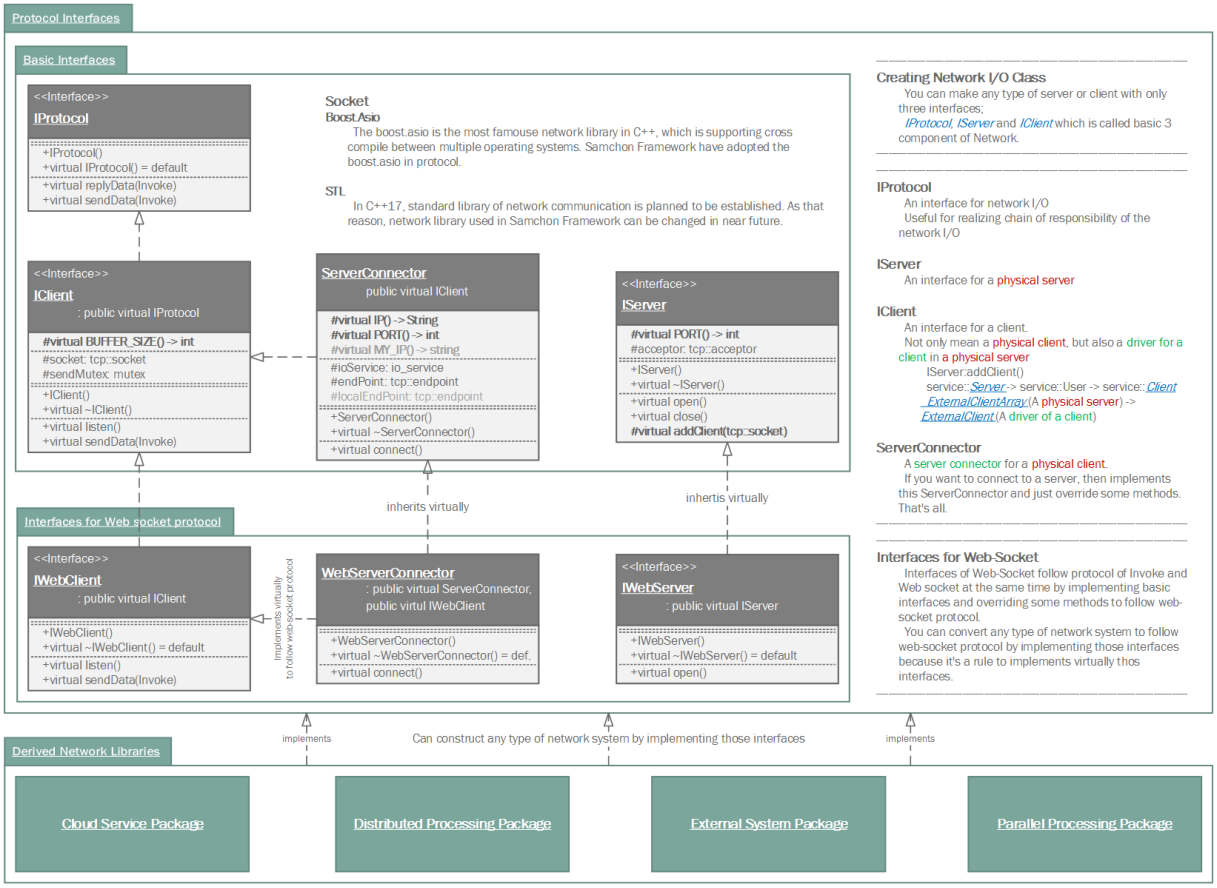
[Inherited]

IClient is an interface not only for a physical client, but also a driver for a client in a physical server.

- IServer:addClient()
  - service::Server -> service::User -> service::Client
  - ExternalClientArray (A physical server) -> ExternalClient (A driver of a client)

IClient is one of the basic 3 + 1 components that can make any type of network system in Samchon Framework with IProtocol and IServer. Looking around classes in Samchon Framework, you can see all classes related with client are implemented from the IClient.

IClient takes a role of real network I/O. IClient sends data to otherside network system from IProtocol::send↩Data(). IClient also gets data from otherside network system, generates standard Invoke message from parsing the replied data and shift the Invoke message to IProtocol::replyData().



Author

Jeongho Nam

6.79.2 Member Function Documentation

6.79.2.1 virtual void listen ( ) [override], [virtual]

Listens message from a related system.

Note

It holds (monopolies) a thread.

Reimplemented from IClient.

6.79.2.2 virtual void sendData (std::shared_ptr< Invoke >) [override], [virtual]

Send Invoke message following web-socket protocol.

Parameters

invoke	Invoke message to send associated network system.

Reimplemented from IClient.

The documentation for this class was generated from the following file:

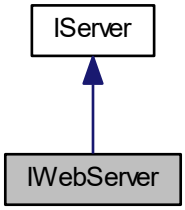
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IWebClient.hpp

6.80 IWebServer Class Reference

An interface for a physical server following web-socket.

```
#include <IWebServer.hpp>
```

Collaboration diagram for IWebServer:



Public Member Functions

- IWebServer ()

  Default Constructor.
- virtual void open () override

  Open the web-socket server.

Additional Inherited Members

6.80.1 Detailed Description

An interface for a physical server following web-socket.

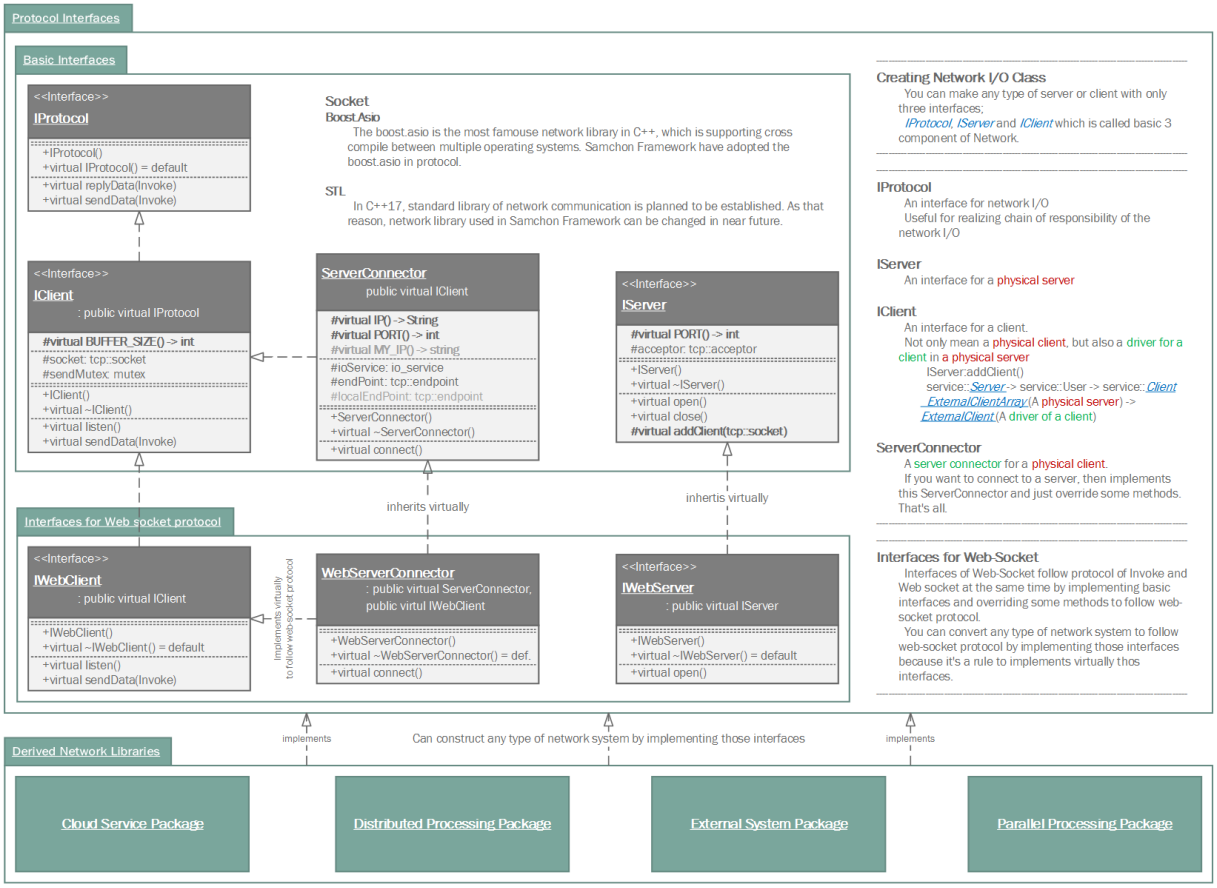
IWebServer is a IServer following web-socket protocol.

[Inherited]

IServer provides methods for opening a server.

IServer is one of the basic 3 + 1 components that can make any type of network system in Samchon Framework with IProtocol and IClient. Looking around classes in Samchon Framework, you can see all servers are implemented from the IServer.

When a client connects to the server, the abstract method IServer::addClient() is called with a new thread. If you want to accept only a client at a time, use OneToOneServer instead.



Example source

A simple chat server running on console

examples/console_chat_server/ChatServer.hpp

```
1 #pragma once
2 #include <samchon/protocol/IServer.hpp>
3 #include <samchon/protocol/IProtocol.hpp>
4
5 #include <set>
6 #include <samchon/library/RWMutex.hpp>
7
8 namespace samchon
9 {
10     namespace example
11     {
21         namespace console_chat_server
22         {
23             class ChatClient;
24
30             class ChatServer
31                 : public protocol::IServer,
32                 public protocol::IProtocol
33             {
34             private:
35                 typedef protocol::IServer super;
36
40                 std::set<ChatClient*> clientSet;
41
45                 library::RWMutex rwMutex;
46
47             public:
51                 ChatServer();
52                 virtual ~ChatServer() = default;
53
57                 void eraseClient(ChatClient *client);
58
59             protected:
63                 virtual auto PORT() const -> int override
64                 {
65                     return 37749;
66                 };
```

```
67
71                  virtual void addClient(protocol::Socket *socket) override;
72
73              public:
80                  virtual void replyData(std::shared_ptr<protocol::Invoke>) override;
81
88                  virtual void sendData(std::shared_ptr<protocol::Invoke>) override;
89              };
90          };
91      };
92 };
```

examples/console_chat_server/ChatServer.cpp

```
1 #include <samchon/examples/console_chat_server/ChatServer.hpp>
2 #include <samchon/examples/console_chat_server/ChatClient.hpp>
3
4 #include <thread>
5 #include <samchon/protocol/Invoke.hpp>
6
7 using namespace std;
8 using namespace samchon::library;
9 using namespace samchon::protocol;
10 using namespace samchon::example::console_chat_server;
11
12 ChatServer::ChatServer()
13     : super()
14 {
15 }
16 void ChatServer::addClient(Socket *socket)
17 {
18     ChatClient *client = new ChatClient(this, socket);
19
20     UniqueWriteLock uk(rwMutex);
21     clientSet.insert(client);
22
23     thread(&ChatClient::listen, client).detach();
24 }
25 void ChatServer::eraseClient(ChatClient *client)
26 {
27     UniqueWriteLock uk(rwMutex);
28     clientSet.erase(client);
29 }
30
31 void ChatServer::replyData(shared_ptr<Invoke> invoke)
32 {
33     sendData(invoke);
34 }
35 void ChatServer::sendData(shared_ptr<Invoke> invoke)
36 {
37     UniqueReadLock uk(rwMutex);
38
39     for(auto it = clientSet.begin(); it != clientSet.end(); it++)
40         (*it)->sendData(invoke);
41 }
```

examples/console_chat_server/ChatClient.hpp

```
1 #pragma once
2 #include <samchon/protocol/IClient.hpp>
3
4 namespace samchon
5 {
6     namespace example
7     {
8         namespace console_chat_server
9         {
10             class ChatServer;
11
21             class ChatClient
22                 : public protocol::IClient
23             {
24             private:
25                 typedef protocol::IClient super;
26
30                 ChatServer *server;
31
32             public:
39                 ChatClient(ChatServer*, protocol::Socket*);
40
45                 virtual ~ChatClient();
46
51                 virtual void replyData(std::shared_ptr<protocol::Invoke>) override;
52             };
53         };
54     };
```

```
};
```

examples/console_chat_server/ChatClient.cpp

```
#include <samchon/examples/console_chat_server/ChatClient.hpp>
#include <samchon/examples/console_chat_server/ChatServer.hpp>

#include <samchon/protocol/Invoke.hpp>

using namespace std;
using namespace samchon::library;
using namespace samchon::protocol;
using namespace samchon::example::console_chat_server;

ChatClient::ChatClient(ChatServer *server, Socket *socket)
    : super()
{
    this->server = server;
    this->socket = socket;
}
ChatClient::~ChatClient()
{
    server->eraseClient(this);
}

void ChatClient::replyData(shared_ptr<Invoke> invoke)
{
    server->replyData(invoke);
}
```

examples/console_chat_server/main.cpp

```
#include <iostream>
#include <samchon/examples/console_chat_server/ChatServer.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

using namespace samchon::example::console_chat_server;

void main()
{
    ChatServer chatServer;
    chatServer.open();

    system("pause");
}
```

Author

Jeongho Nam

6.80.2 Member Function Documentation

6.80.2.1 virtual void open ( ) [override],[virtual]

Open the web-socket server.

Accepts clients and handshake with them following web-socket protocol.

Note

It monopolies a thread.

Reimplemented from IServer.

The documentation for this class was generated from the following file:

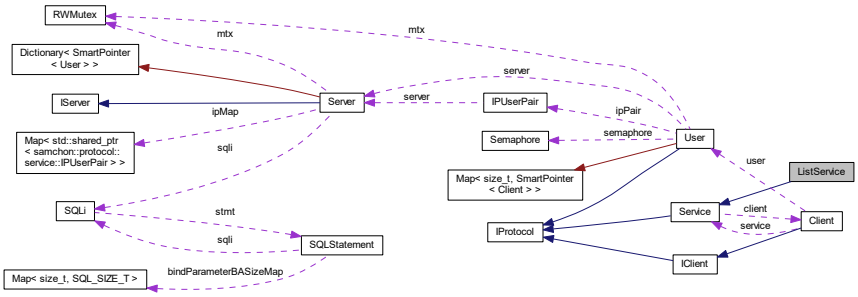
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/IWebServer.hpp

6.81 ListService Class Reference

A list service of chat rooms.

```
#include <ListService.hpp>
```

Collaboration diagram for ListService:



Public Member Functions

- ListService (protocol::service::Client *)

  Construct from a client.

Private Member Functions

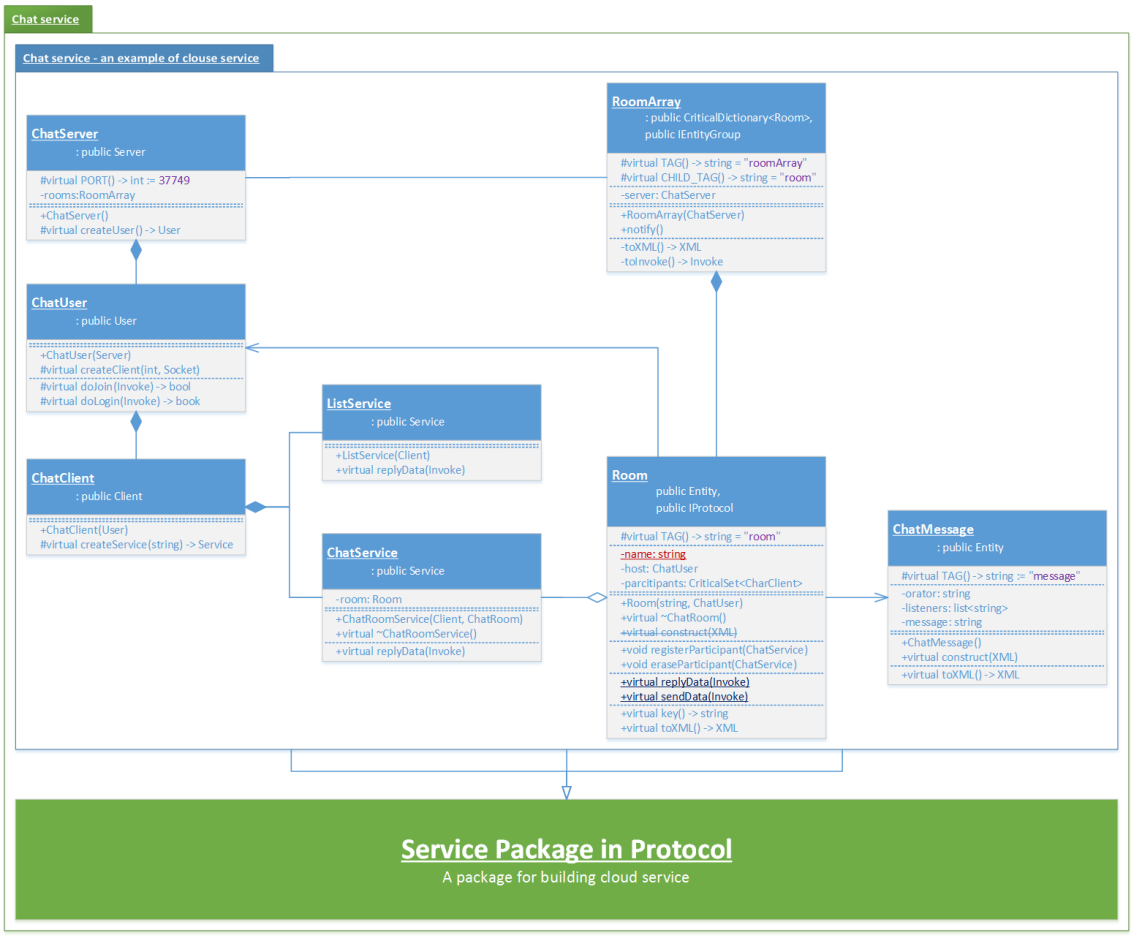
- void createRoom (const std::string &)

  Create a chat room.

Additional Inherited Members

6.81.1 Detailed Description

A list service of chat rooms.



Author

Jeongho Nam

6.81.2 Member Function Documentation

6.81.2.1 void createRoom (const std::string &) [private]

Create a chat room.

Parameters

name	Name of the candidate room to create.

The documentation for this class was generated from the following file:

– D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/chat_service/ListService.hpp

6.82 Map< _Kty, _Ty, _Pr, _Alloc > Class Template Reference

Map.

```
#include <Map.hpp>
```

Public Member Functions

– auto has (const _Kty &key) const -> bool

Whether have the item or not.

- auto get (const _Kty &key) -> _Ty &
  Get element.
- void set (const _Kty &key, const _Ty &val)
  Set element.
- auto pop (const _Kty &key) -> _Ty
  Pop item.

6.82.1 Detailed Description

template<typename _Kty, typename _Ty, typename _Pr = std::less<_Kty>, typename _Alloc = std::allocator<std::pair<const _Kty, _Ty>>>class samchon::Map< _Kty, _Ty, _Pr, _Alloc >

Map.

Map is a std::map some methods are modified

- Addicted methods
  - has(find(key) != end()) method is addicted
  - at method is addicted
  - set method is addicted
  - pop method is addicted
- Modified methods
  - at was depreciated (get is different with at)

[Inherited]

Maps are associative containers that store elements formed by a combination of a key value and a mapped value, following a specific order.

In a map, the key values are generally used to sort and uniquely identify the elements, while the mapped values store the content associated to this key. The types of key and mapped value may differ, and are grouped together in member type value_type, which is a pair type combining both:

- typedef pair<const Key, T> value_type;

Internally, the elements in a map are always sorted by its key following a specific strict weak ordering criterion indicated by its internal comparison object (of type Compare).

map containers are generally slower than unordered_map containers to access individual elements by their key, but they allow the direct iteration on subsets based on their order.

Maps are typically implemented as binary search trees.

The mapped values in a map can be accessed directly by their corresponding key using the bracket operator ((operator[]).

Maps are typically implemented as binary search trees.

Referenced comments of std::map

- http://www.cplusplus.com/reference/map/map/

Template Parameters

_Kty	

Type of the keys. Each element in a map is uniquely identified by its key value.

Aliased as member type map::key_type.

Template Parameters

_Ty	

Type of the mapped value. Each element in a map stores some data as its mapped value.

Aliased as member type map::mapped_type.

Template Parameters

_Pr	

A binary predicate that takes two element keys as arguments and returns a bool. The expression comp(a,b), where comp is an object of this type and a and b are key values, shall return true if a is considered to go before b in the strict weak ordering the function defines.

The map object uses this expression to determine both the order the elements follow in the container and whether two element keys are equivalent (by comparing them reflexively: they are equivalent if !comp(a,b) && !comp(b,a)). No two elements in a map container can have equivalent keys.

This can be a function pointer or a function object. This defaults to std::less<T>, which returns the same as applying the less-than operator (a<b).

Aliased as member type map::key_compare.

Template Parameters

_Alloc	

Type of the allocator object used to define the storage allocation model. By default, the allocator class template is used, which defines the simplest memory allocation model and is value-independent.

Aliased as member type map::allocator_type.

Author

Jeongho Nam

6.82.2 Member Function Documentation

6.82.2.1 auto has (const _Kty & *key*) const -> bool [inline]

Whether have the item or not.

Indicates whether a map has an item having the specified identifier.

Parameters

key	Key value of the element whose mapped value is accessed.

Returns

Whether the map has an item having the specified identifier

6.82.2.2 auto get (const _Kty & *key*) -> _Ty& [inline]

Get element.

Returns a reference to the mapped value of the element identified with key

Warning

get is different with std::map's at. get does not create object but throws excention if the matched key doesn't exist.

Parameters

key	Key value of the element whose mapped value is accessed.

Exceptions

exception	out of range

Returns

A reference object of the mapped value (_Ty)

6.82.2.3 void set (const _Kty & *key*, const _Ty & *val*) [inline]

Set element.

Set an item as the specified identifier.

If the identifier is already in map, change value of the identifier.
If not, then insert the object with the identifier.

Parameters

key	Key value of the element whose mapped value is accessed.
val	Value, the item.

6.82.2.4 auto pop (const _Kty & *key*) -> _Ty [inline]

Pop item.

Removes an item having specified key and returns the removed element.

Returns

An item released by pop

The documentation for this class was generated from the following file:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/Map.hpp

6.83 Math Class Reference

Utility of Math.

```
#include <Math.hpp>
```

Static Public Member Functions

- static auto random () -> double
  Get a random value.
- static auto degree_to_radian (double) -> double
  Convert degree to radian.
- static auto radian_to_degree (double) -> double
  Convert radian to degree.
- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto minimum (const _Cont &container) -> IndexPair< _Ty >
  Calculate minimum value with its index.
- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto maximum (const _Cont &container) -> IndexPair< _Ty >
  Calculate maximum value with its index.

- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto mean (const _Cont &container) -> double

  Calculate average.
- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto median (const _Cont &container) -> double

  Get median value.
- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto mode (const _Cont &container) -> _Ty

  Find mode value.
- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto stdev_p (const _Cont &container) -> double

  Square root of a variance_p.
- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto stdev_s (const _Cont &container) -> double

  Square root of a variance_s.
- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto variance_p (const _Cont &container) -> double

  Calculate standard deviation for all population.
- template<typename _Cont , typename _Ty = _Cont::value_type>
  static auto variance_s (const _Cont &container) -> double

  Calculate variance for sample.

Static Public Attributes

- static const double E = exp(1.0)

  Exponent.
- static const double PI = 3.141592653589793

  mber Pi
- static const double LN2 = 1.0 / log2(E)

  $log_e 2$
- static const double LN10 = 1.0 / log10(E)

  $log_e 10$
- static const double LOG2E = log2(E)

  $log_2 e$
- static const double LOG10E = log10(E)

  $log_{10} e$
- static const double SQRT1_2 = sqrt(.5)

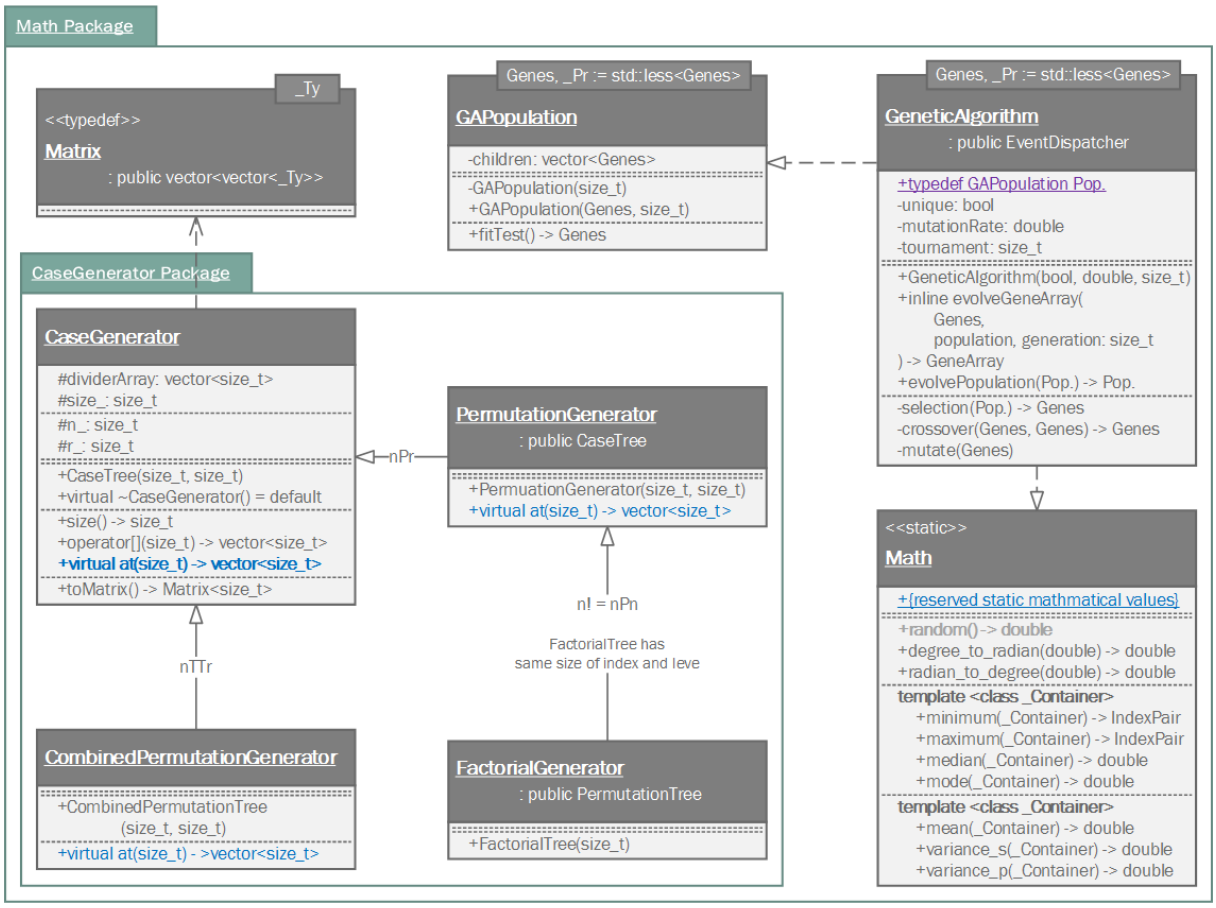
  Squart root of 0.5.
- static const double SQRT2 = sqrt(2.0)

  Squart root of 2.

6.83.1 Detailed Description

Utility of Math.

Provides some static values and mathmatical methods.



See also

samchon::library

Author

Jeongho Nam

6.83.2 Member Function Documentation

6.83.2.1 auto random ( ) -> double [static]

Get a random value.

Returns a random value following uniform distribution

Returns

The random value

6.83.2.2 static auto minimum (const _Cont & *container*) -> IndexPair<_Ty> [inline],[static]

Calculate minimum value with its index.

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its minimum value

Returns

Pair of minimum value and its index

6.83.2.3 static auto maximum (const _Cont & *container*) -> IndexPair<_Ty> [inline],[static]

Calculate maximum value with its index.

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its minimum value

Returns

Pair of maximum value and its index

6.83.2.4 static auto mean (const _Cont & *container*) -> double [inline],[static]

Calculate average.

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its mean

Returns

Average(mean) of the contaier

6.83.2.5 static auto median (const _Cont & *container*) -> double [inline],[static]

Get median value.

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its median

Returns

Median value of the container

6.83.2.6 static auto mode (const _Cont & *container*) -> _Ty [inline],[static]

Find mode value.

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its mode

Returns

Most appeared value in the container

6.83.2.7 static auto stdev_p (const _Cont & *container*) -> double [inline],[static]

Square root of a variance_p.

$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its standard deviation

Returns

Standard deviation of the container

6.83.2.8 static auto stdev_s (const _Cont & *container*) -> double [inline],[static]

Square root of a variance_s.

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its variance

Returns

Variance of the container

6.83.2.9 static auto variance_p (const _Cont & *container*) -> double [inline], [static]

Calculate standard deviation for all population.

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its standard deviation

Returns

Standard deviation of the container

6.83.2.10 static auto variance_s (const _Cont & *container*) -> double [inline], [static]

Calculate variance for sample.

Template Parameters

_Cont	Type of the container
_Ty	

Value type of the container.

Do not modify the default template argument.

Parameters

container	Container to find its variance

Returns

Variance of the container

6.83.3 Member Data Documentation

6.83.3.1 const double E = exp(1.0) [static]

Exponent.

A mathematical constant for the base of natural logarithms, expressed as e. The approximate value of e is 2.71828182845905.

6.83.3.2 const double PI = 3.141592653589793 [static]

mber Pi

A mathematical constant for the ratio of the circumference of a circle to its diameter, expressed as pi, with a value of 3.141592653589793.

6.83.3.3 const double LN2 = 1.0 / log2(E) [static]

$\log_e 2$

A mathematical constant for the natural logarithm of 2, expressed as $\log_e 2$, with an approximate value of 0.6931471805599453.

6.83.3.4 const double LN10 = 1.0 / log10(E) [static]

$\log_e 10$

A mathematical constant for the natural logarithm of 10, expressed as $\log_e 10$, with an approximate value of 2.302585092994046.

6.83.3.5 const double LOG2E = log2(E) [static]

$\log_2 e$

A mathematical constant for the base-2 logarithm of the constant e, expressed as $\log_2 e$, with an approximate value of 1.442695040888963387.

6.83.3.6 const double LOG10E = log10(E) [static]

$\log_{10} e$

A mathematical constant for the base-10 logarithm of the constant e (Math.E), expressed as $\log_{10} e$, with an approximate value of 0.4342944819032518. The Math.log() method computes the natural logarithm of a number. Multiply the result of Math.log() by Math.LOG10E to obtain the base-10 logarithm.

6.83.3.7 const double SQRT1_2 = sqrt(.5) [static]

Squart root of 0.5.

A mathematical constant for the square root of one-half, with an approximate value of 0.7071067811865476.

6.83.3.8 const double SQRT2 = sqrt(2.0) [static]

Squart root of 2.

A mathematical constant for the square root of 2, with an approximate value of 1.4142135623730951.

The documentation for this class was generated from the following files:

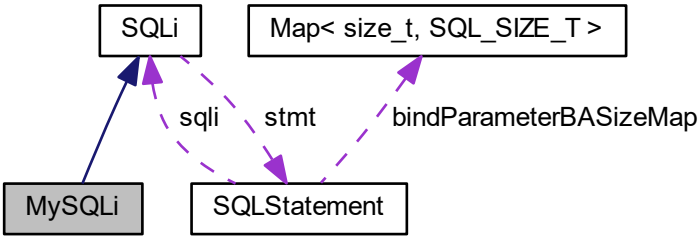
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Math.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Math.cpp

6.84 MySQLi Class Reference

A SQL interface for My-SQL.

```
#include <MySQLi.hpp>
```

Collaboration diagram for MySQLi:



Public Member Functions

- MySQLi (int port=3306)

  Construct from port.

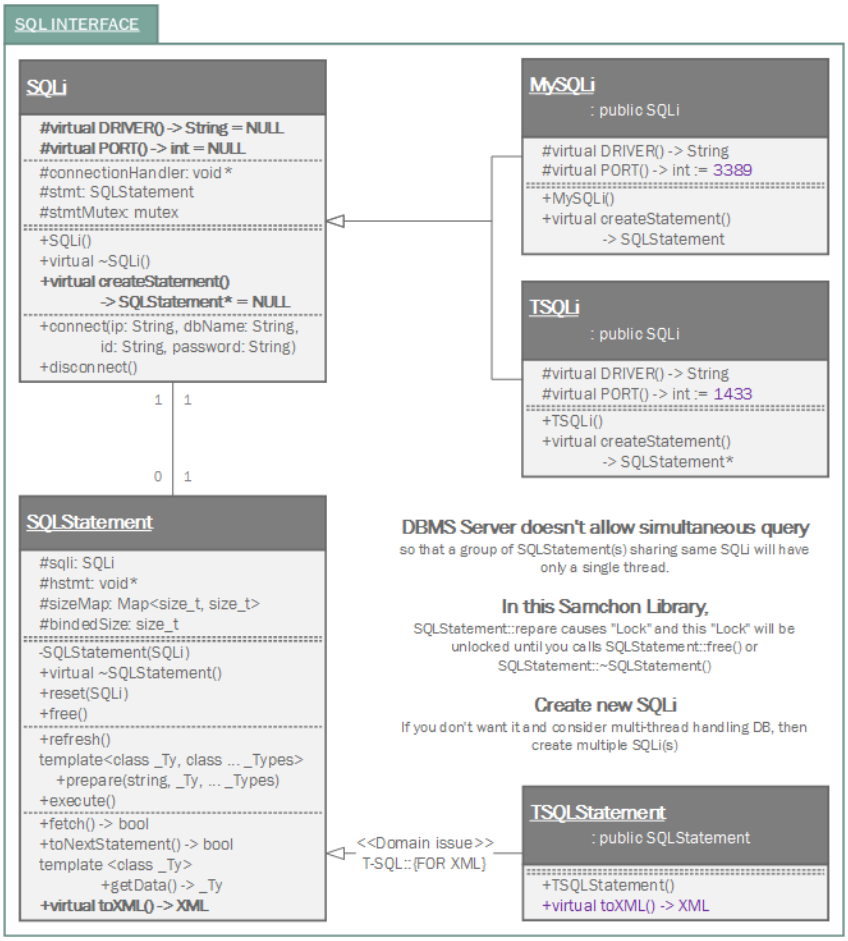
Additional Inherited Members

6.84.1 Detailed Description

A SQL interface for My-SQL.

SQLi is an abstract class managing connection (session) with a specific database and manages SQL statements are executed and results are returned within the context of a connection.

The reason why SQLStatement(s) are managed by SQLi is because of a principle of DBMS, DBMS system doesn't allow simultaneous query from a session (connection). Only a query (process0) is allowed at a time. If you try to simultaneous query from a SQLi, ODBC throws error. To avoid the error, an execute of query from SQLStatement will lock a mutex of SQLi to ensure exclusiveness.



Note

To ensure the exclusiveness, you've to make SQLStatement from SQLi. Do not make SQLStatement by yourself. call SQLi::createSQLStatement() instead. Even you make a derived class from SQLStatement, don't make its constructor to have public accessor.

Becuase execution of a sql statement causes lock on mutex in SQLi, you've to destruct the SQLStatement or call SQLStatement::free() method(). If you don't, the mutex will not be unlocked, thus you can't do anything by the SQLi.

See also

samchon::library

Author

Jeongho Nam

6.84.2 Constructor & Destructor Documentation

6.84.2.1 MySQLi (int *port* = 3306)

Construct from port.

Parameters

port	Port number of My-SQL. default is 3306.

The documentation for this class was generated from the following file:

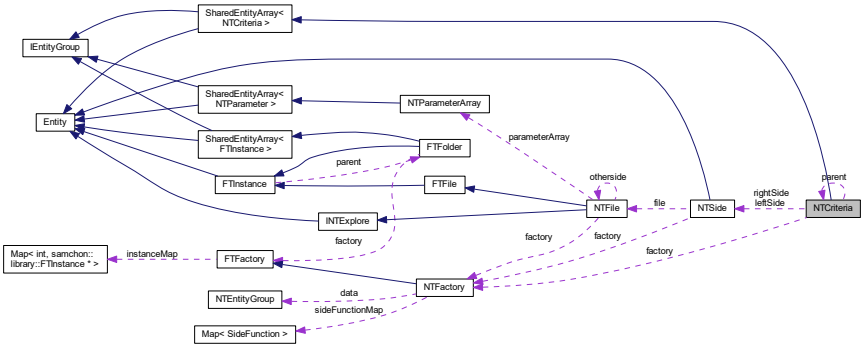
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/MySQLi.hpp

6.85 NTCriteria Class Reference

Criteria, a conditional expression with weight.

`#include <NTCriteria.hpp>`

Collaboration diagram for NTCriteria:



Public Member Functions

- virtual void construct (std::shared_ptr< library::XML >) override
  Construct data of the Entity from an XML object.
- virtual auto TAG () const -> std::string override
  A tag name when represented by XML.
- virtual auto CHILD_TAG () const -> std::string override
  A tag name of children.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the EntityGroup.

Protected Member Functions

- virtual auto createChild (std::shared_ptr< library::XML >) -> NTCriteria *override
  Factory method of a child Entity.

6.85.1 Detailed Description

Criteria, a conditional expression with weight.

NTCriteria is an object realizing ANN(Artificial Neural Network).

You can make the ANN model having weight and bias.

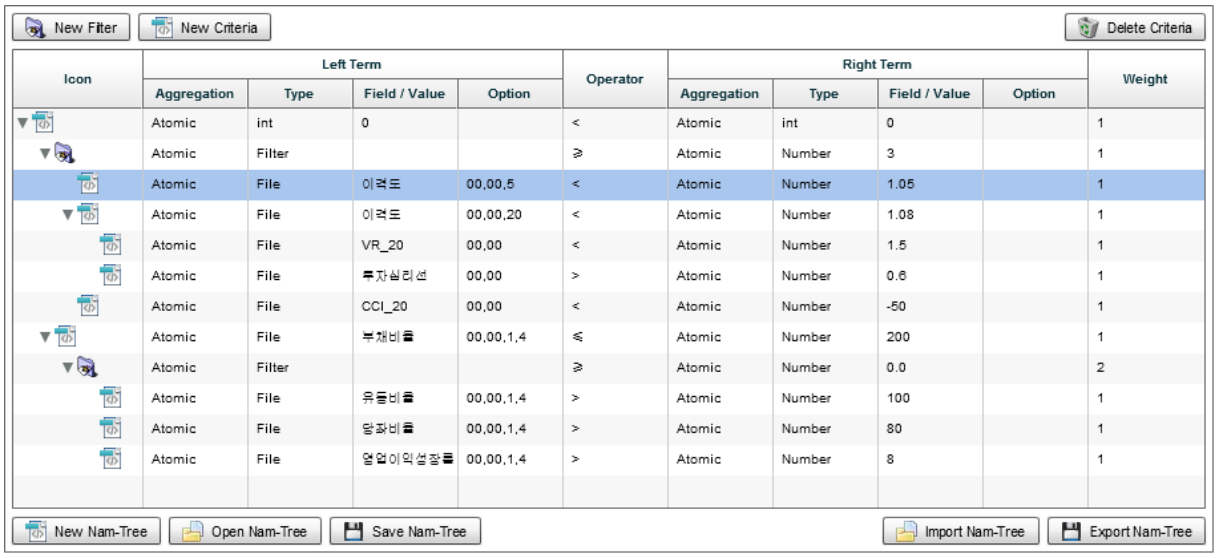
1. A conditional expression with weight
   F(x) = (NTSide < NTSide ? 1: 0) X weight
   NTCrietria is made up for conditional expression.
   - When the expression is true, returns 1 * weight

- When the expression is false, returns 0

2. Hierarchical relationship

   - In vertical relationship: multiply (X)
   - In horizontal relationship: plus (+)

   With that rule, you can make enormous conditions. I can sure there's not any condition can'be be expressed by the model.



Icon	Left Term				Operator	Right Term				Weight
	Aggregation	Type	Field / Value	Option		Aggregation	Type	Field / Value	Option	
	Atomic	int	0		<	Atomic	int	0		1
	Atomic	Filter			≥	Atomic	Number	3		1
	Atomic	File	이격도	00,00,5	<	Atomic	Number	1.05		1
	Atomic	File	이격도	00,00,20	<	Atomic	Number	1.08		1
	Atomic	File	VR_20	00,00	<	Atomic	Number	1.5		1
	Atomic	File	투자심리선	00,00	>	Atomic	Number	0.6		1
	Atomic	File	CCI_20	00,00	<	Atomic	Number	-50		1
	Atomic	File	부채비율	00,00,1,4	≤	Atomic	Number	200		1
	Atomic	Filter			≥	Atomic	Number	0.0		2
	Atomic	File	유동비율	00,00,1,4	>	Atomic	Number	100		1
	Atomic	File	당좌비율	00,00,1,4	>	Atomic	Number	80		1
	Atomic	File	영업이익성장률	00,00,1,4	>	Atomic	Number	8		1

3. Making bias

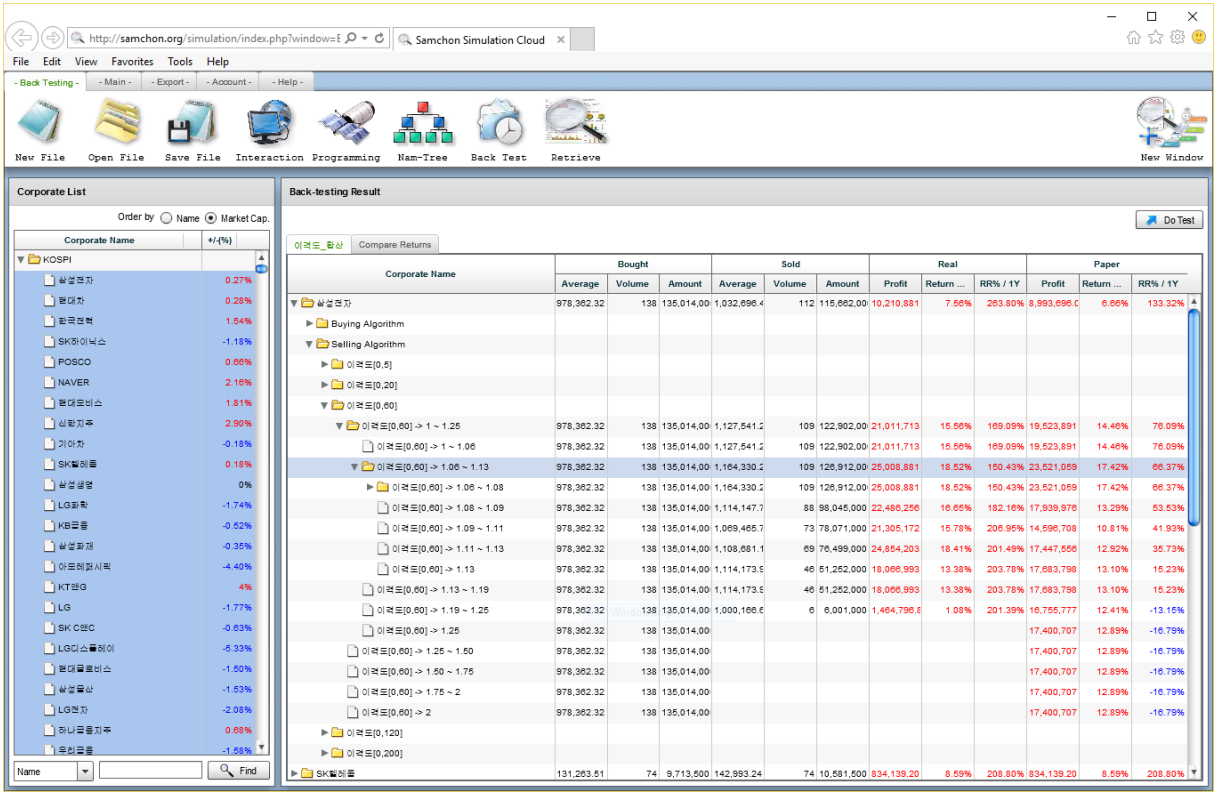
   Just make a NTCriteria returns only true.

   It's the bias returns only weight.

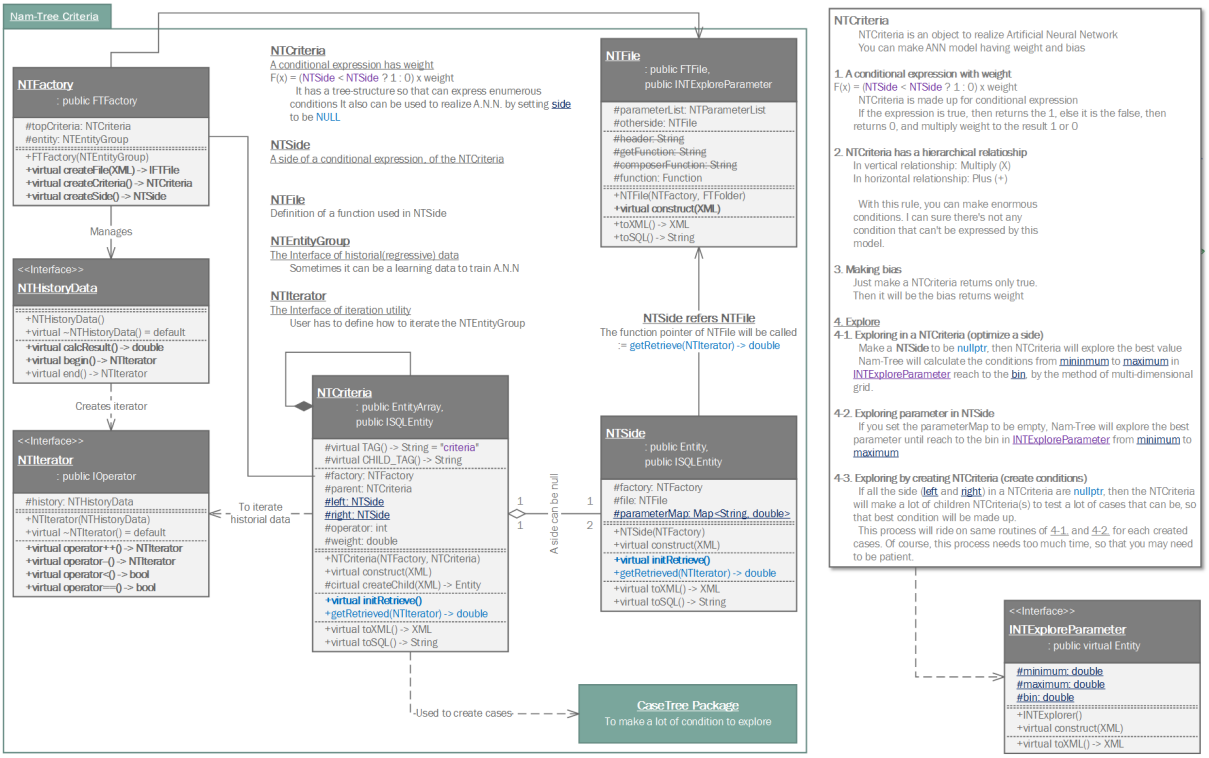
4. Explore

   (a) Exporing in a NTCriteria (optimize a side)
   Make a NTSide to be nullptr, then NTCriteria will explore the best value.
   Nam-Tree will calculate the condtions from minimum to maximum in INTExplore reach to the precision, by the method of multi-demensional grid.

   (b) Exploring parameter in NTSide
   If you set the parameterMNap to be empty, Nam-Tree will explore the best parameter until reach to the precision in INTExploreParameter from minimum to maximum.



(c) Exploring by creating NTCriteria
(Creates lots of hierarchical conditions and tests them)
If all the side (left and right) in a NTCriteria are nullptr, the NTCriteria will make a lot of children NTCriteria(s) to test lots of cases, so that derives the best condition set will be made up.
This process will ride on same routines of 4-1 and 4-2, for each created cases. Of course, this process needs too much time, so that you may need to be patient.



See also

samchon::namtree

Author

Jeongho Nam

6.85.2 Member Function Documentation

6.85.2.1 void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

6.85.2.2 auto createChild (std::shared_ptr< library::XML >) -> NTCriteria *override [override], [protected], [virtual]

Factory method of a child Entity.

EntityGroup::createChild() is a factory method creating a new child Entity which is belonged to the Entity↩ Group. This method is called by EntityGroup::construct(). The children construction methods Entity↩ ::construct() will be called by abstract method of the EntityGroup::construct().

Returns

A new child Entity belongs to EntityGroup.

Implements EntityGroup< _Container, _ETy, _Ty >.

6.85.2.3 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.85.2.4 auto CHILD_TAG ( ) const -> std::string [override], [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

6.85.2.5 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTCriteria.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTCriteria.cpp

6.86 NTEntityGroup Class Reference

A historical, studying data.

```
#include <NTEntityGroup.hpp>
```

Public Member Functions

- NTEntityGroup ()

  Default Constructor.
- virtual auto begin () const -> std::shared_ptr< NTIterator >=0

  Return iterator to beginning.
- virtual auto end () const -> std::shared_ptr< NTIterator >=0

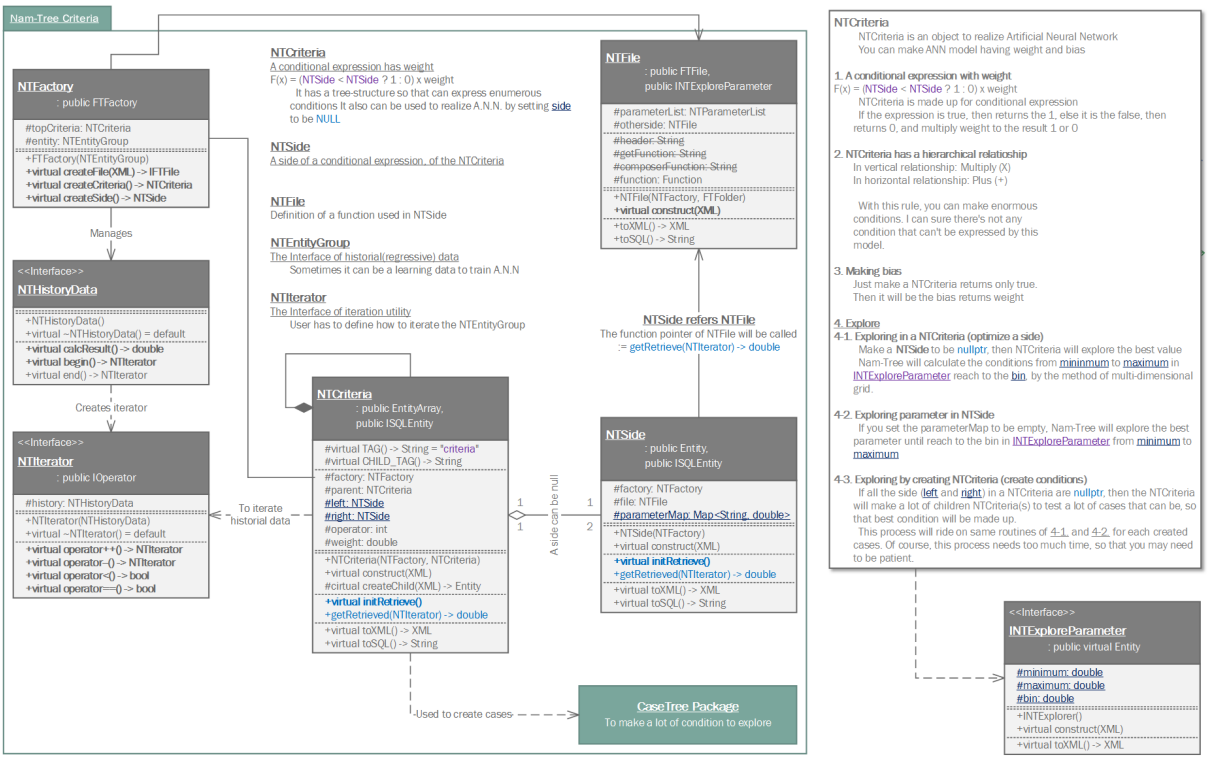
  Return iterator to end.

6.86.1 Detailed Description

A historical, studying data.

NTEntityGroup is an abstract class representing historical, regressive data

The NTEntitGroup is used to studying data evolving and training NTCriteria, representing Artificial neural network.



See also

samchon::namtree

Author

Jeongho Nam

6.86.2 Member Function Documentation

6.86.2.1 virtual auto begin ( ) const -> std::shared_ptr< NTIterator > [pure virtual]

Return iterator to beginning.

Returns an iterator referring the first element in the historical data container.

Note

If the container is empty, the returned iterator is same with end().

Returns

An iterator to the first element in the container.

The iterator containes the first element's pair; key and value.

6.86.2.2 virtual auto end ( ) const -> std::shared_ptr< NTIterator > [pure virtual]

Return iterator to end.

Returns an iterator referring to the past-the-end element in the historical data container.

The past-the-end element is the theoretical element that would follow the last element in the NTEntityGroup container. It does not point to any element, and thus shall not be dereferenced.

Because the ranges used by functions of the NTEntityGroup do not include the element reference by their closing iterator, this function is often used in combination with NTEntityGroup::begin() to specify a range including all the elements in the container.

Note

Returned iterator from NTEntityGroup::end() does not refer any element. Trying to accessing element by the iterator will cause throwing exception (out of range).

If the container is empty, this function returns the same as NTEntityGroup::begin().

The documentation for this class was generated from the following files:

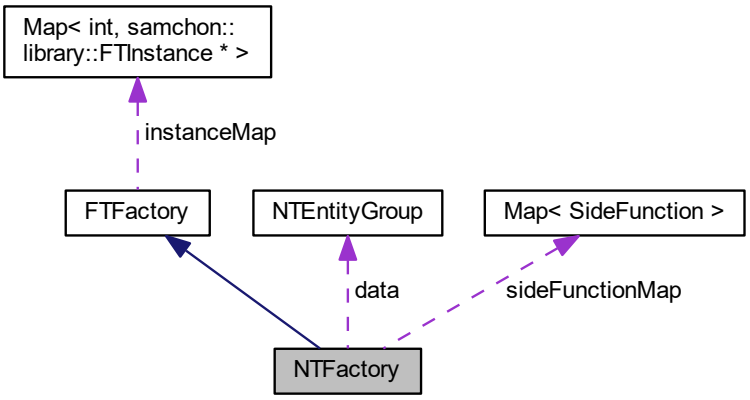
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTEntityGroup.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTEntityGroup.cpp

6.87 NTFactory Class Reference

A factory for Nam-Tree objects.

`#include <NTFactory.hpp>`

Collaboration diagram for NTFactory:



Public Member Functions

- NTFactory (NTEntityGroup ∗)

  Construct from historical(studying) data.
- virtual auto createCriteria (NTCriteria ∗, std::shared_ptr< library::XML >) -> NTCriteria ∗

  Factory method of a NTCriteria.
- virtual auto createSide (std::shared_ptr< library::XML >) -> NTSide ∗

  Factory method of a NTSide.

Protected Attributes

- NTEntityGroup ∗ data

  A entity gorup containing historical data.
- Dictionary< SideFunction > sideFunctionMap

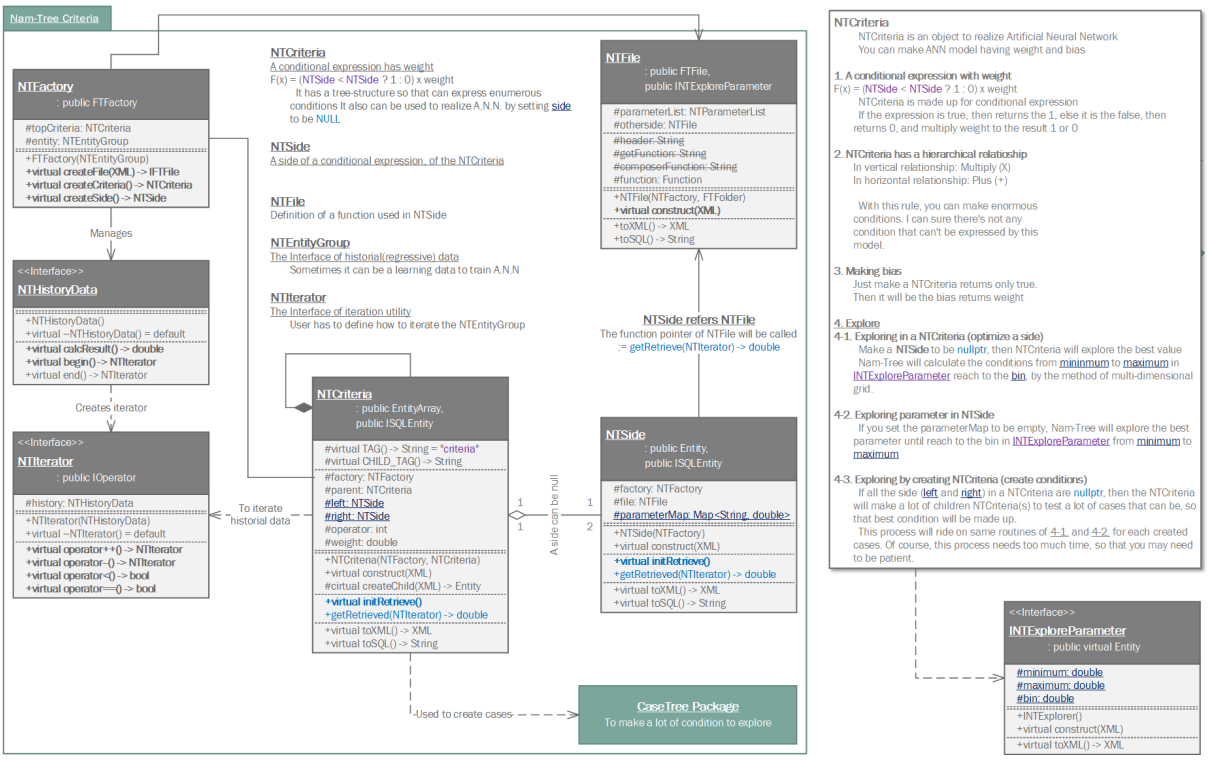
  A dictionary of function pointers.

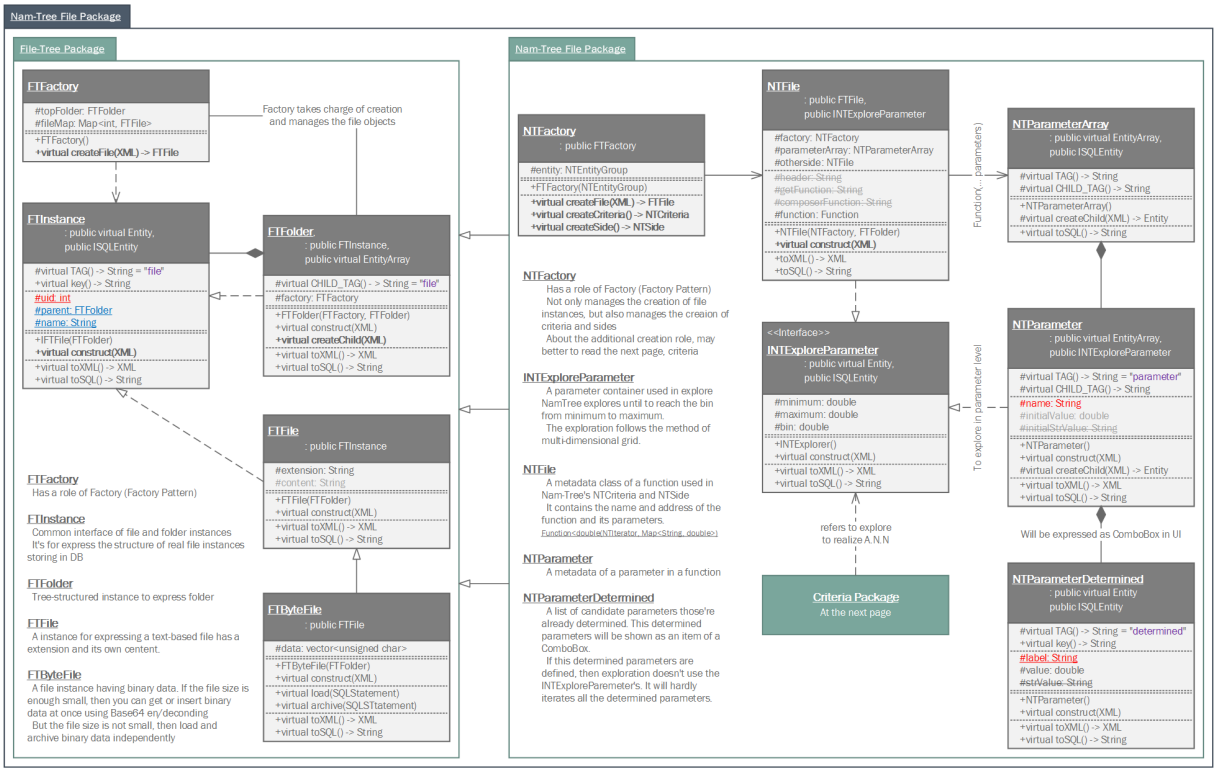
6.87.1 Detailed Description

A factory for Nam-Tree objects.

NTFactory class is a FTFactory and an abstract class creating and managing Nam-Tree objects. When overriding nam-tree module, do not override methods *createChild()* of file or criteria objects like NTFile, NT↩Criteria and NTSide. Only overrides method of creation in the NTFactory.

– Creator method of history data is not in NTFactory. It must be done by hand of user.





Note

Previous version of the Samchon Framework, NTFile class had a function script on the NTFile and Nam-Tree module compiled the script like Nam-Tree moudle of Flex standalone that is keeping the compiling method.

However, modern Samchon Framework's C++ NTFile class doesn't have a function script and does not compile. I don't know a way to compile a script in lots of operating systems, so I dropped the compiling method. Until update, NTFactory has function pointers in a Dictionary and NTFile reference them. Users must put own methods into the Dictionary by their hands.

I will solve the problem sooon. Until next generation of Samchon Framework, put your own function pointers to a Dictionary in NTFactory by your hand please. Sorry for my unripe skill on programming.

See also

samchon::namtree

Author

Jeongho Nam

6.87.2 Constructor & Destructor Documentation

6.87.2.1 NTFactory (NTEntityGroup * *data*)

Construct from historical(studying) data.

Parameters

entity	A group of history data.

6.87.3 Member Function Documentation

6.87.3.1 auto createCriteria (NTCriteria *, std::shared_ptr< library::XML >) -> NTCriteria* [virtual]

Factory method of a NTCriteria.

Parameters

parent	A parent NTCriteria object xml An xml object for discriminating type.

Returns

A NTCrteria object representing a logical conditional.

6.87.3.2 auto createSide (std::shared_ptr< library::XML >) -> NTSide* [virtual]

Factory method of a NTSide.

Parameters

xml	An xml oject for distriminating type.

Returns

A NTSide object which means a side of a logical condition.

The documentation for this class was generated from the following files:

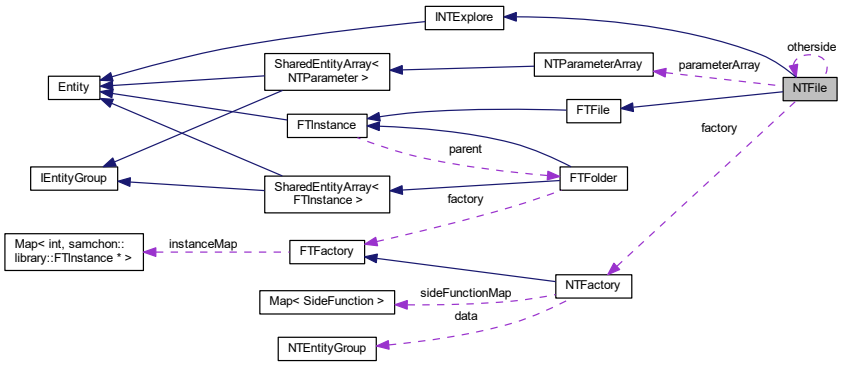
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTFactory.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTFactory.cpp

6.88 NTFile Class Reference

A file archiving metadata of a function.

`#include <NTFile.hpp>`

Collaboration diagram for NTFile:



Public Member Functions

- NTFile (NTFactory ∗, library::FTFolder ∗)

  Construct from factory and parent folder.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- auto getParameterArray () const -> NTParameterArray ∗

  Get metadata of parameters.
- auto getOtherside () const -> NTFile ∗

  Get otherside file.
- auto getFunction () const -> SideFunction

  Get function pointer.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the Entity.

Protected Attributes

- NTFactory ∗ factory

  A factory and manager class for nam-tree.
- NTParameterArray ∗ parameterArray

  Metadata of parameters.
- NTFile ∗ otherside

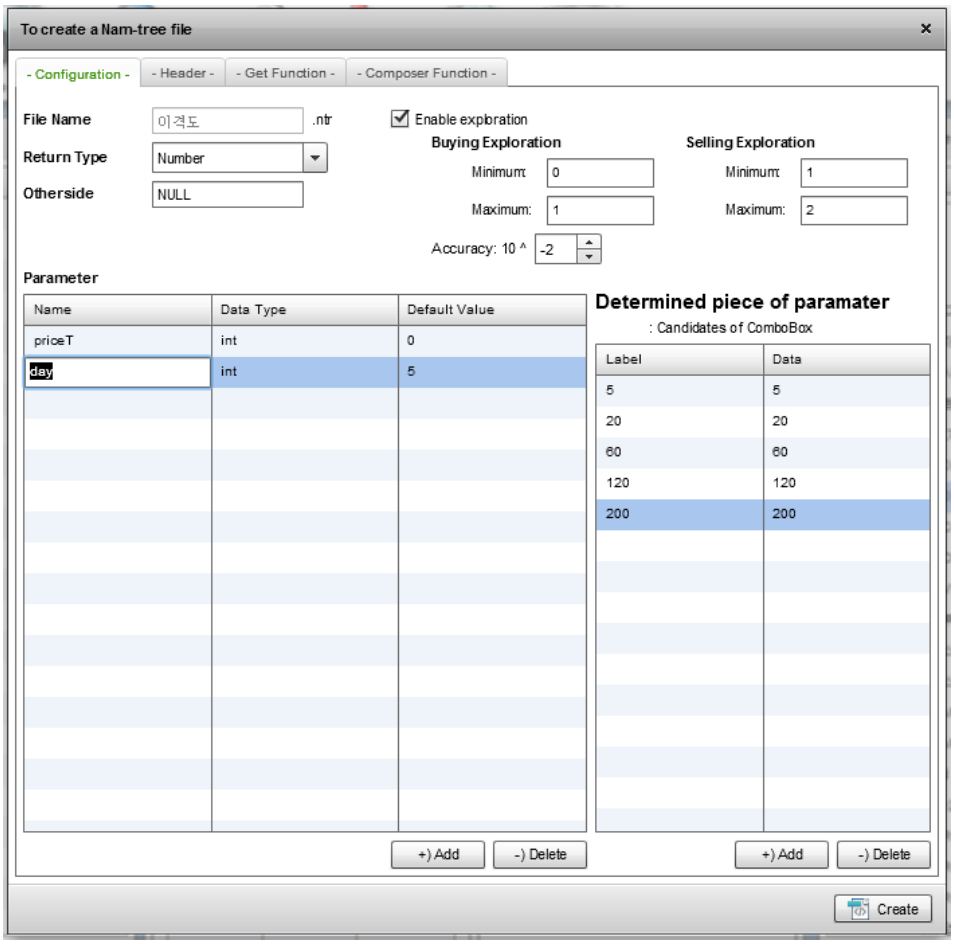
  A file of otherside.
- SideFunction function

  A function pointer.

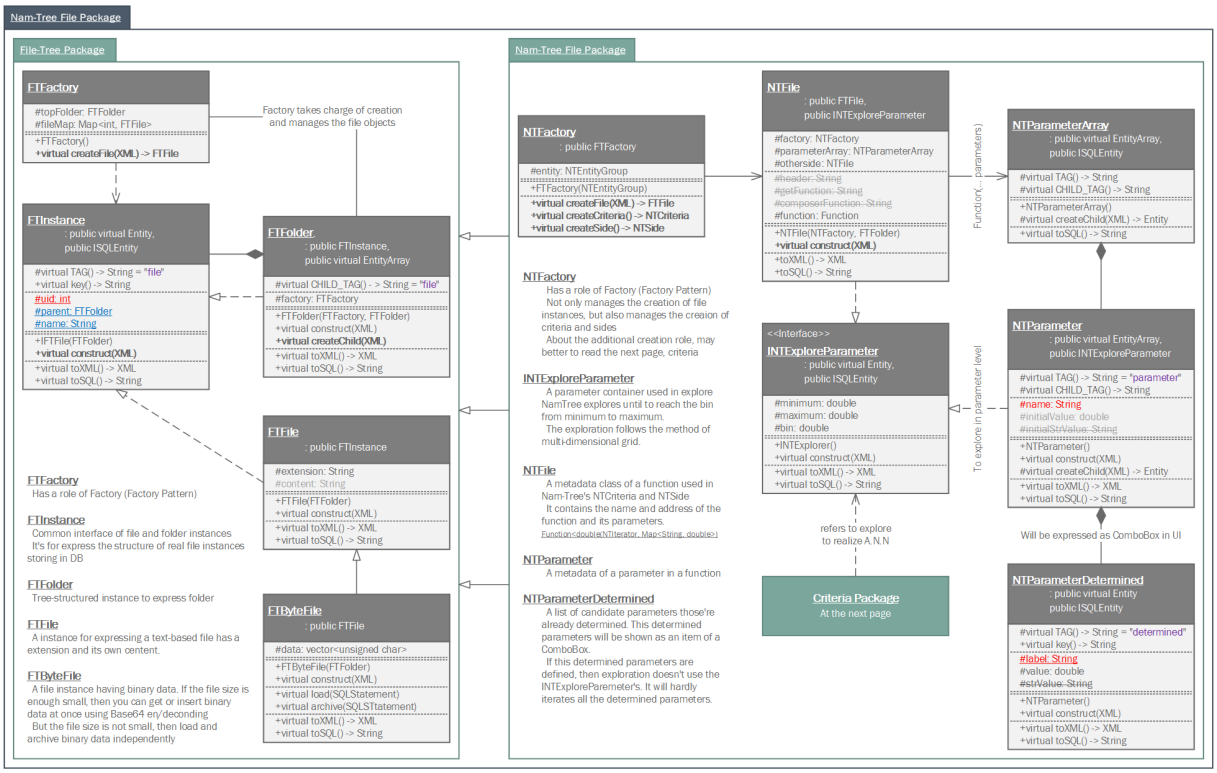
Additional Inherited Members

6.88.1 Detailed Description

A file archiving metadata of a function.



NTFile class is a FTFile archiving meta of a function which is used by a NTSide class. It may better to understand what NTFile is, if you consider the NTFile as a header of a function. Name, metadata of parameters and function pointer is archived on the NTFile.



Note

Previous version of the Samchon Framework, NTFile class had a function script on the NTFile and Nam-Tree module compiled the script like Nam-Tree moudle of Flex standalone that is keeping the compiling method.

However, modern Samchon Framework's C++ NTFile class doesn't have a function script and does not compile. I don't know a way to compile a script in lots of operating systems, so I dropped the compiling method. Until update, NTFactory has function pointers in a Dictionary and NTFile reference them. Users must put own methods into the Dictionary by their hands.

I will solve the problem sooon. Until next generation of Samchon Framework, put your own function pointers to a Dictionary in NTFactory by your hand please. Sorry for my unripe skill on programming.

See also

samchon::namtree
namtree::NTSide

Author

Jeongho Nam

6.88.2 Constructor & Destructor Documentation

6.88.2.1 NTFile (NTFactory * *factory*, library::FTFolder * *parent*)

Construct from factory and parent folder.

Parameters

factory	A factory class creating nam-tree objects.

folder	A parent folder of the NTFile.

6.88.3 Member Function Documentation

6.88.3.1 void construct (std::shared_ptr< library::XML >) [override], [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from INTExplore.

6.88.3.2 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from FTFile.

6.88.4 Member Data Documentation

6.88.4.1 NTFactory* factory [protected]

A factory and manager class for nam-tree.

NTFile references NTFactory to find otherside file and function pointer.

6.88.4.2 NTFile* otherside [protected]

A file of otherside.

The variable otherside is a pair of file which is inserted to a criteria's opposite side.

When the otherside is specified and you determine the NTFile to be contained in a NTSide, the NTCriteria owning the NTSide will determine opposite side's NTSide to have the otherside NTFile forcibly.

Note

The determination will be automatic and unchangeble. Consider hardly when specifying an otherside.

The documentation for this class was generated from the following files:

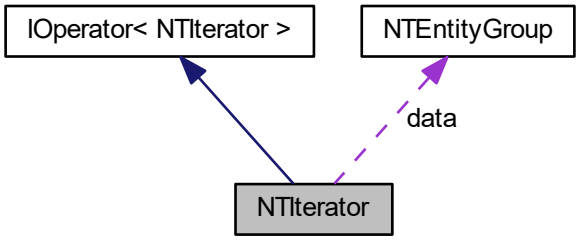
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTFile.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTFile.cpp

6.89 NTIterator Class Reference

Iterator of historical data.

`#include <NTIterator.hpp>`

Collaboration diagram for NTIterator:



Public Member Functions

- NTIterator (const NTEntityGroup *)

  Construt from historical data.
- virtual auto operator-- () const -> std::shared_ptr< NTIterator >=0

  Get iterator to previous element.
- virtual auto operator++ () const -> std::shared_ptr< NTIterator >=0

  Get iterator to next element.
- virtual auto operator== (const NTIterator &) const -> bool=0

  Whether an iterator is equal with the iterator.
- virtual auto operator< (const NTIterator &) const -> bool=0

  Whether the iterator is less than an iterator.

Protected Attributes

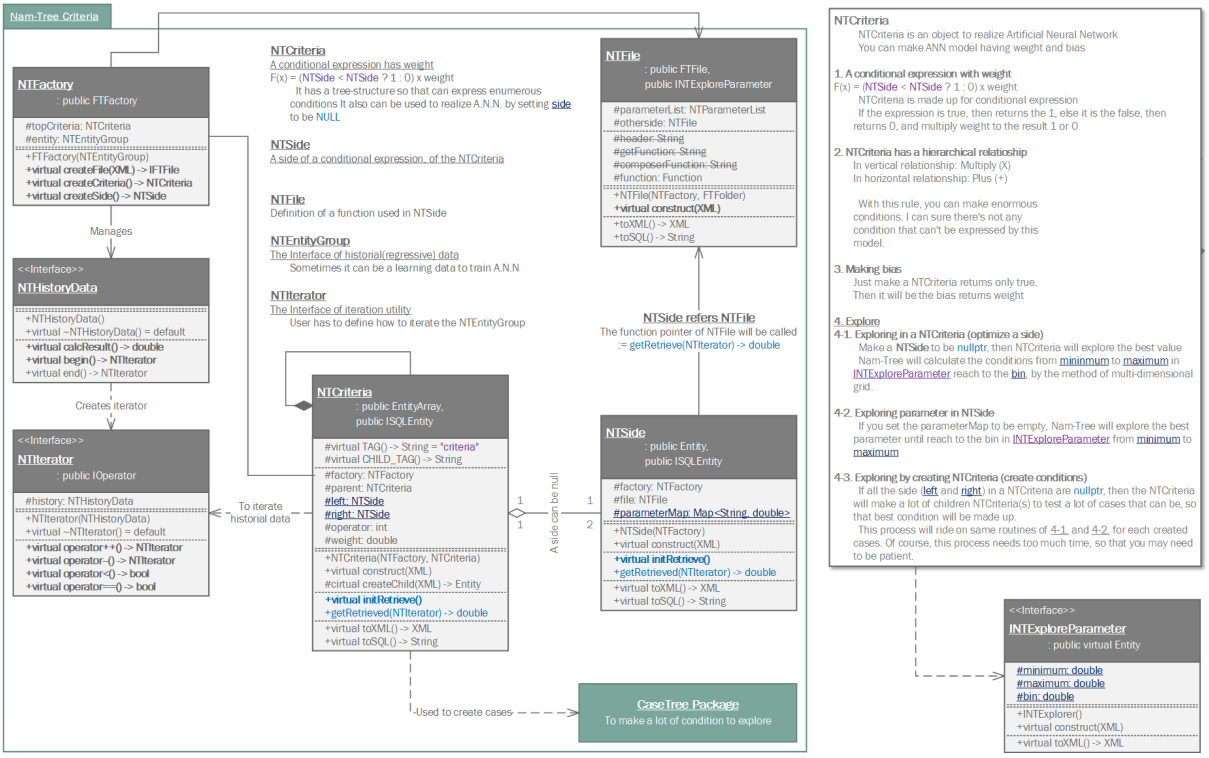
- const NTEntityGroup * data

  A historical data, source of the iterator.

6.89.1 Detailed Description

Iterator of historical data.

NTIterator is an abstract iterator of NTEntityGroup.



See also

samchon::namtree

Author

Jeongho Nam

6.89.2 Member Function Documentation

6.89.2.1 virtual auto operator-- ( ) const -> std::shared_ptr< NTIterator > [pure virtual]

Get iterator to previous element.

If current iterator is the first item(equal with *begin()*), returns end().

Returns

An iterator of the previous item.

6.89.2.2 virtual auto operator++ ( ) const -> std::shared_ptr< NTIterator > [pure virtual]

Get iterator to next element.

If current iterator is the last item, returns end().

Returns

An iterator of the next item.

6.89.2.3 virtual auto operator== (const NTIterator &) const -> bool [pure virtual]

Whether an iterator is equal with the iterator.

Compare two iterators and returns whether they are equal or not.

Parameters

obj	An iterator to compare

Returns

Indicates whether equal or not.

6.89.2.4 virtual auto operator< (const NTIterator &) const -> bool [pure virtual]

Whether the iterator is less than an iterator.

Compare two iterators and returns whether this iterator is less.

Parameters

obj	An iterator to compare

Returns

Indicates whether less or not.

The documentation for this class was generated from the following files:

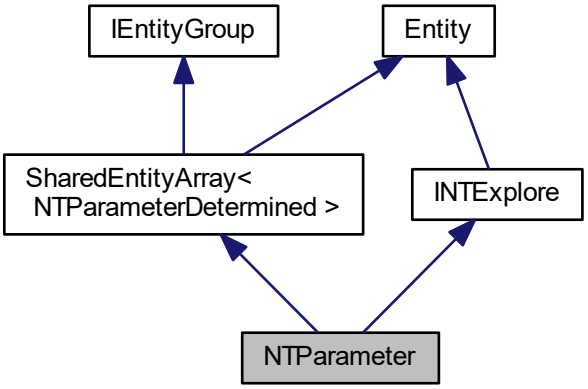
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTIterator.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTIterator.cpp

6.90 NTParameter Class Reference

A metadata of a parameter in a function.

`#include <NTParameter.hpp>`

Collaboration diagram for NTParameter:



Public Member Functions

- NTParameter ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

Construct data of the Entity from an XML object.

- virtual auto key () const -> std::string override

  Get a key that can identify the Entity uniquely.
- auto getName () const -> std::string

  Get name.
- auto getInitialValue () const -> double

  Get initial, default vlaue.
- virtual auto TAG () const -> std::string override

  A tag name when represented by XML.
- virtual auto CHILD_TAG () const -> std::string override

  A tag name of children.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the Entity.

Protected Member Functions

- virtual auto createChild (std::shared_ptr< library::XML >) -> NTParameterDetermined *override

  Factory method of a child Entity.

Protected Attributes

- std::string name

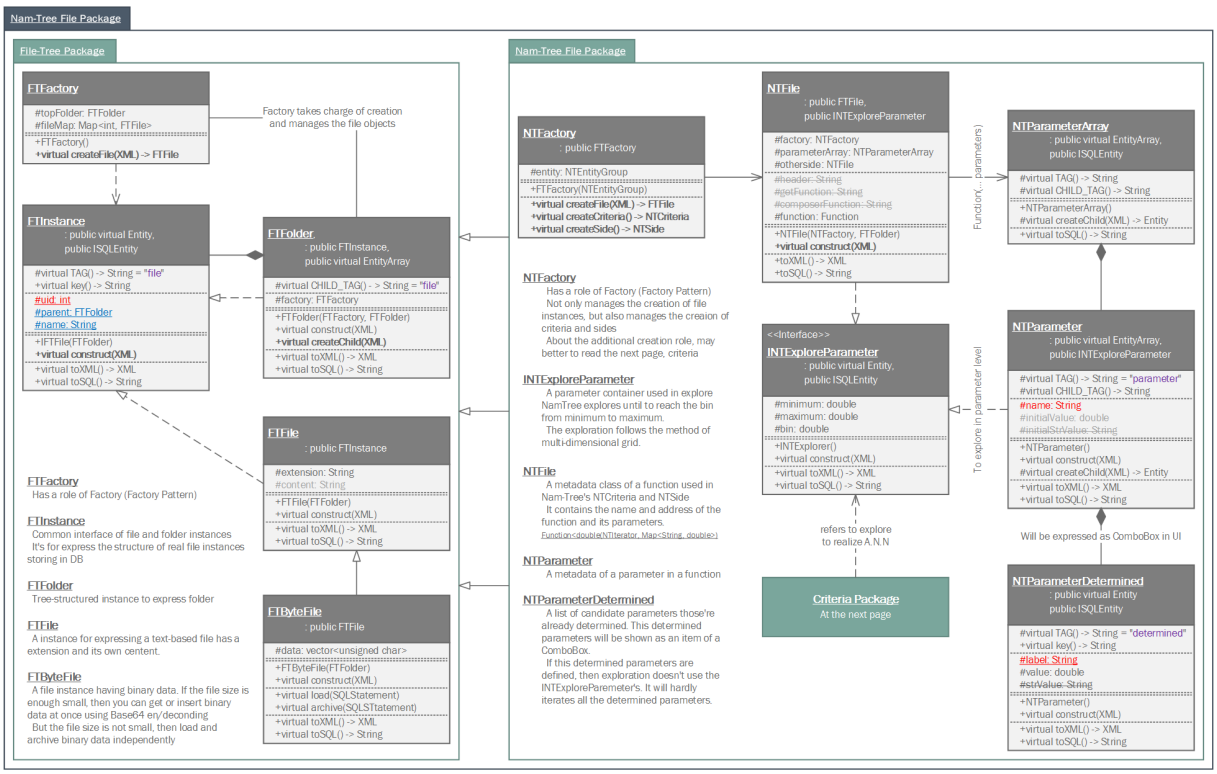
  Name of a paremeter.
- double initialValue

  An initial, default value.

6.90.1 Detailed Description

A metadata of a parameter in a function.

NTParameter is a class containing metadata of a parameter which is belonged to a class, metadata of a function, NTFile. Between the NTFile and NTParameter(s), there's a associate relationship.



See also

samchon::namtree

Author

Jeongho Nam

6.90.2 Member Function Documentation

6.90.2.1 void construct (std::shared_ptr< library::XML >) [override], [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from INTExplore.

6.90.2.2 auto createChild (std::shared_ptr< library::XML >) -> NTParameterDetermined *override [override], [protected], [virtual]

Factory method of a child Entity.

EntityGroup::createChild() is a factory method creating a new child Entity which is belonged to the Entity↩ Group. This method is called by EntityGroup::construct(). The children construction methods Entity↩ ::construct() will be called by abstract method of the EntityGroup::construct().

Returns

A new child Entity belongs to EntityGroup.

Implements EntityGroup< _Container, _ETy, _Ty >.

6.90.2.3 auto key ( ) const -> std::string [override], [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented from Entity.

6.90.2.4 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.90.2.5 auto CHILD_TAG ( ) const -> std::string [override], [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

6.90.2.6 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from INTExplore.

6.90.3 Member Data Documentation

6.90.3.1 std::string name [protected]

Name of a paremeter.

The string label is identifier of a NTParameter. It must be unique.

The documentation for this class was generated from the following files:

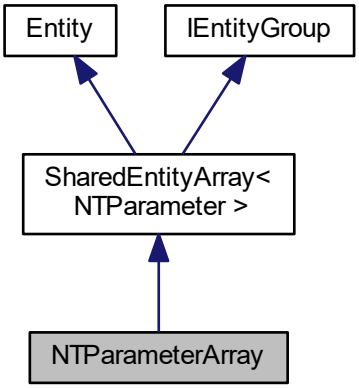
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTParameter.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTParameter.cpp

6.91 NTParameterArray Class Reference

An Array of NTParameter.

```
#include <NTParameterArray.hpp>
```

Collaboration diagram for NTParameterArray:



Public Member Functions

- virtual auto TAG () const -> std::string override

  A tag name when represented by XML.
- virtual auto CHILD_TAG () const -> std::string override

  A tag name of children.
- NTParameterArray ()

  Default Constructor.

Protected Member Functions

- virtual auto createChild (std::shared_ptr< library::XML >) -> NTParameter *override

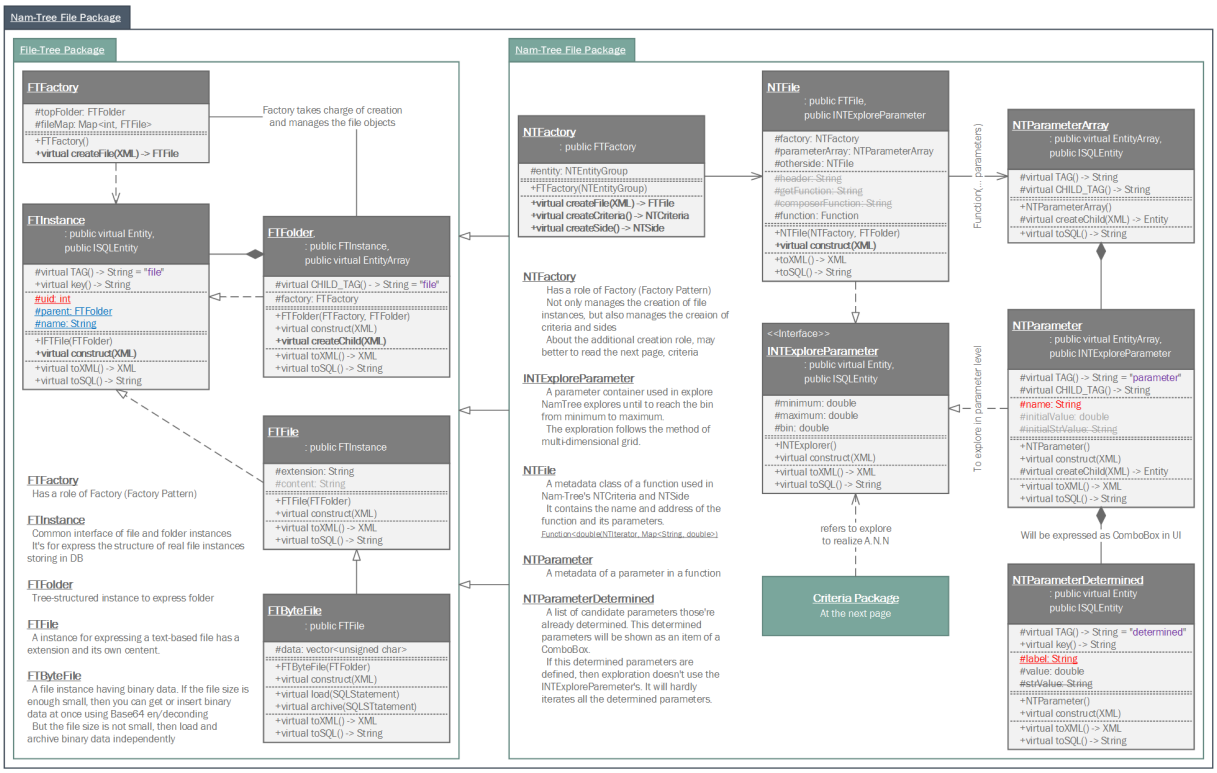
  Factory method of a child Entity.

6.91.1 Detailed Description

An Array of NTParameter.

NTParameterArray is an array class containing NTParameter objects.

The NTParameter is belonged to a NTFile class to realize associate relationship between NTFile and NT↩ Parameter(s).



See also

samchon::namtree

Author

Jeongho Nam

6.91.2 Member Function Documentation

6.91.2.1 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.91.2.2 auto CHILD_TAG ( ) const -> std::string [override], [virtual]

A tag name of children.

< TAG>
<CHILD_TAG />
<CHILD_TAG />
</TAG>

Implements IEntityGroup.

6.91.2.3 auto createChild (std::shared_ptr< library::XML >) -> NTParameter *override [override], [protected], [virtual]

Factory method of a child Entity.

EntityGroup::createChild() is a factory method creating a new child Entity which is belonged to the Entity↩ Group. This method is called by EntityGroup::construct(). The children construction methods Entity↩ ::construct() will be called by abstract method of the EntityGroup::construct().

Returns

A new child Entity belongs to EntityGroup.

Implements EntityGroup< _Container, _ETy, _Ty >.

The documentation for this class was generated from the following files:

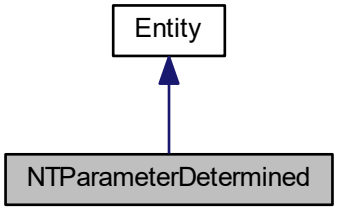
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTParameterArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTParameterArray.cpp

6.92 NTParameterDetermined Class Reference

A pre-determined value of a parameter.

`#include <NTParameterDetermined.hpp>`

Collaboration diagram for NTParameterDetermined:



Public Member Functions

- NTParameterDetermined ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual auto key () const -> std::string override

  Get a key that can identify the Entity uniquely.
- auto getLabel () const -> std::string

  Get label.
- auto getValue () const -> double

  Get value.
- virtual auto TAG () const -> std::string override

  A tag name when represented by XML.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

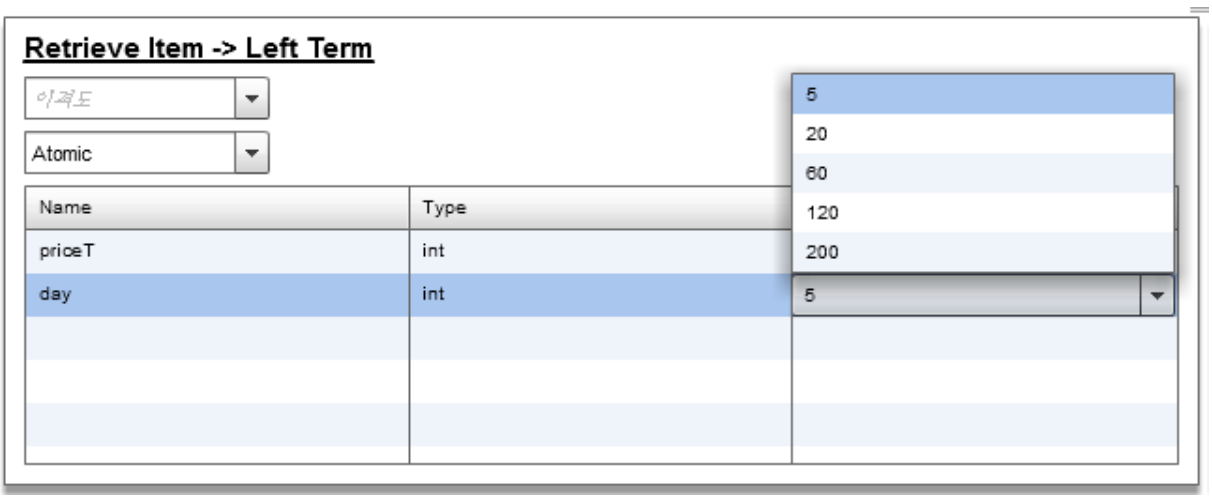
  Get an XML object represents the Entity.

Protected Attributes

- std::string label
  A label will be printed in a combo-box.
- double value
  A value, pre-defined.

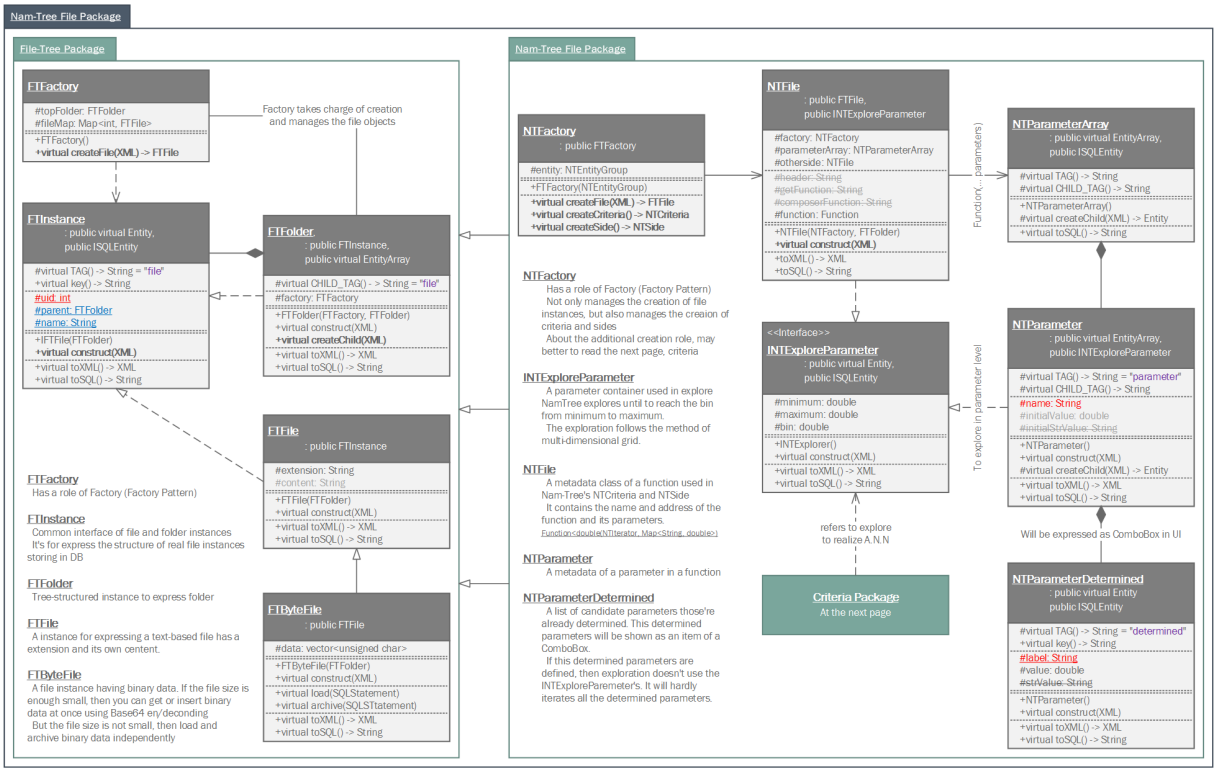
6.92.1 Detailed Description

A pre-determined value of a parameter.



NTParameterDetermined is a class belonged to a NTParameter object presents a pre-defined value. In UI module of Nam-Tree, the NTParameterDetermined is expressed by an item of a ComboBox.

When optimization of Nam-Tree, in a part of NTParameter, if the pre-determined value is defined optimizer does not explore with grid method from referencing INTExplore. Just selects a value between the pre-defineds. (One of NTParameterDetermined is selected in NTParameter).



See also

samchon::namtree

Author

Jeongho Nam

6.92.2 Member Function Documentation

6.92.2.1 void construct (std::shared_ptr< library::XML >) [override], [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

6.92.2.2 auto key ( ) const -> std::string [override], [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented from Entity.

6.92.2.3 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.92.2.4 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

6.92.3 Member Data Documentation

6.92.3.1 std::string label [protected]

A label will be printed in a combo-box.

The string label is identifier of a NTParameterDetermined. It must be unique.

The documentation for this class was generated from the following files:

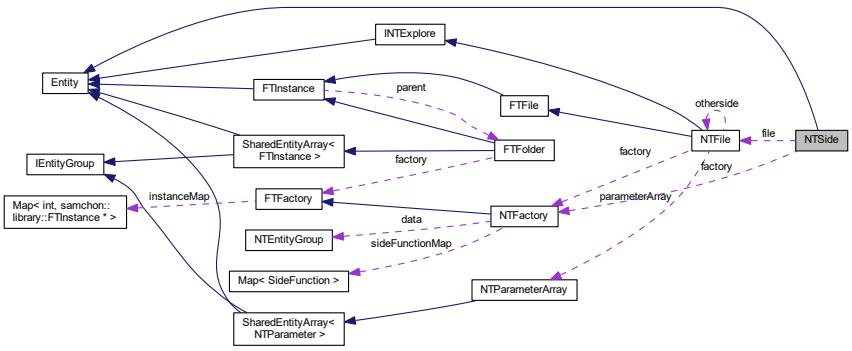
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTParameterDetermined.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTParameterDetermined.cpp

6.93 NTSide Class Reference

A side of a conditional expresson.

`#include <NTSide.hpp>`

Collaboration diagram for NTSide:



Public Member Functions

- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the Entity.

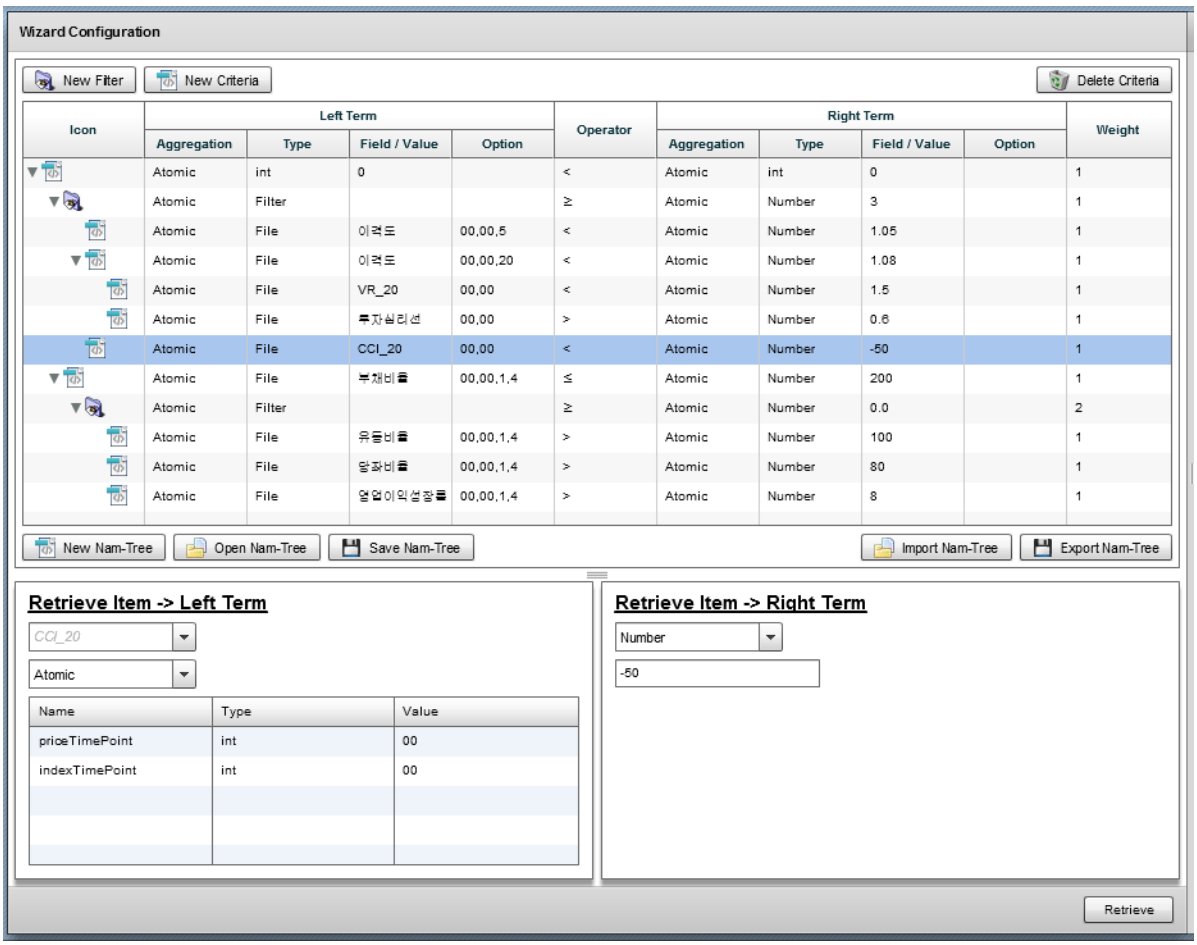
Protected Member Functions

- virtual auto TAG () const -> std::string override

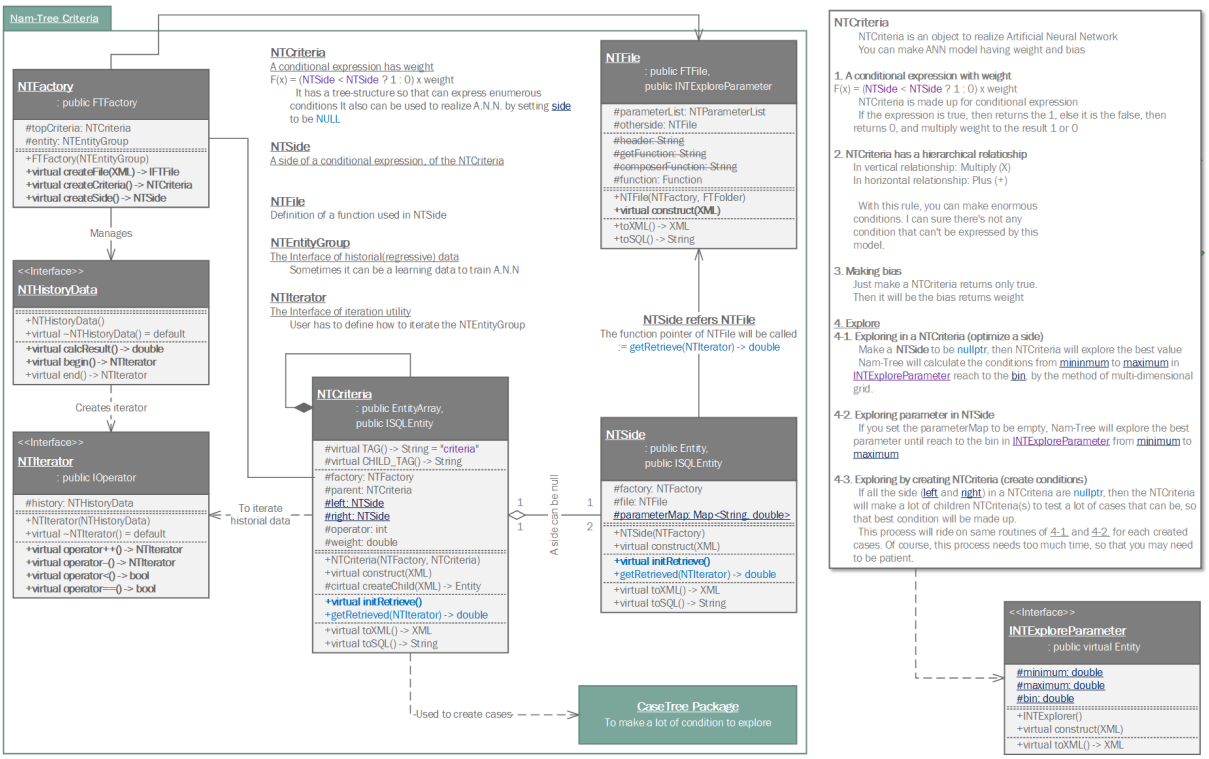
  A tag name when represented by XML.

6.93.1 Detailed Description

A side of a conditional expresson.



Two NTSide classes are belonged to a NTCriteria as left and right side. The NTSide class has a NTFile representing a function and parameters to adjust to the function.



See also

samchon::namtree

Author

Jeongho Nam

6.93.2 Member Function Documentation

6.93.2.1 auto TAG ( ) const -> std::string [override], [protected], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.93.2.2 void construct (std::shared_ptr< library::XML >) [override], [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

6.93.2.3 auto toXML ( ) const -> std::shared_ptr<library::XML> [override],[virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

The documentation for this class was generated from the following files:

- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTSide.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/namtree/NTSide.cpp

6.94 Packer Class Reference

A packer planning the best packaging.

```
#include <Packer.hpp>
```

Public Member Functions

- Packer (std::vector< Product > *, std::vector< Wrapper > *)

  Construct from products and wrapper.
- Packer (const Packer &)

  Copy Constructor.
- void optimize ()

  Find the best packaging method.
- auto calcPrice () const -> int

  Calculate price of the wrappers.
- auto toString () const -> std::string

  Return a string represents an packaging method.

Private Attributes

- std::vector< Product > * productArray

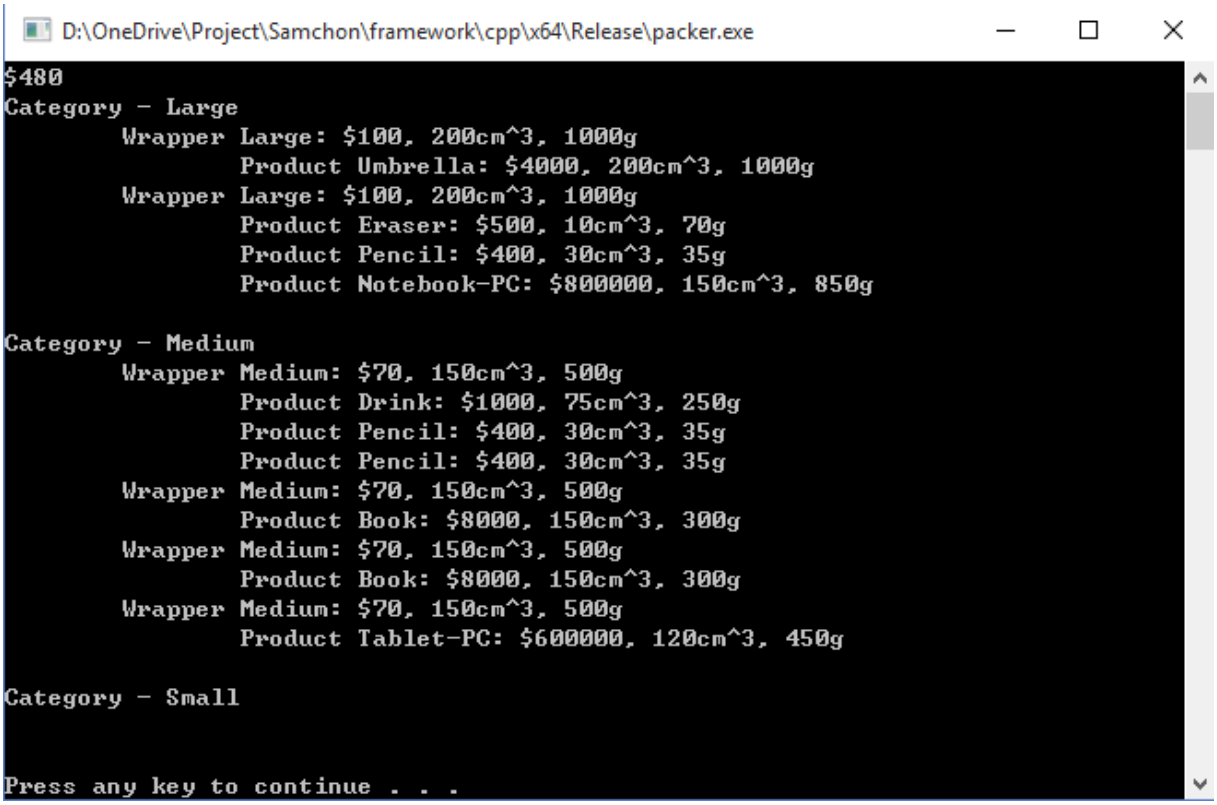
  Prodcut(s) to package in some Wrapper(s)
- std::vector< Wrapper > * wrapperArray

  Type of Wrapper(s) to be used for packaging.

6.94.1 Detailed Description

A packer planning the best packaging.

Retrieves the solution of packaging by combination permuation and factorial case.



Warning

Be careful about number of products and wrappers.

The time complexity of Packer overs O(m^n). Elapsed time of calculation increases enourmously. Do not use Packer if the digits of number of products or wrappers overs 2.

Author

Jeongho Nam

6.94.2 Constructor & Destructor Documentation

6.94.2.1 Packer (std::vector< Product > * *productArray*, std::vector< Wrapper > * *wrapperArray*)

Construct from products and wrapper.

Parameters

productArray	Product(s) to input some Wrapper
wrapperArray	Type of Wrapper(s) to be used

6.94.2.2 Packer (const Packer & *packer*)

Copy Constructor.

Copy constructor of Packer does not copy children items.

Only copies member variables(productArray, wrapperArray) of Packer's.

The documentation for this class was generated from the following files:

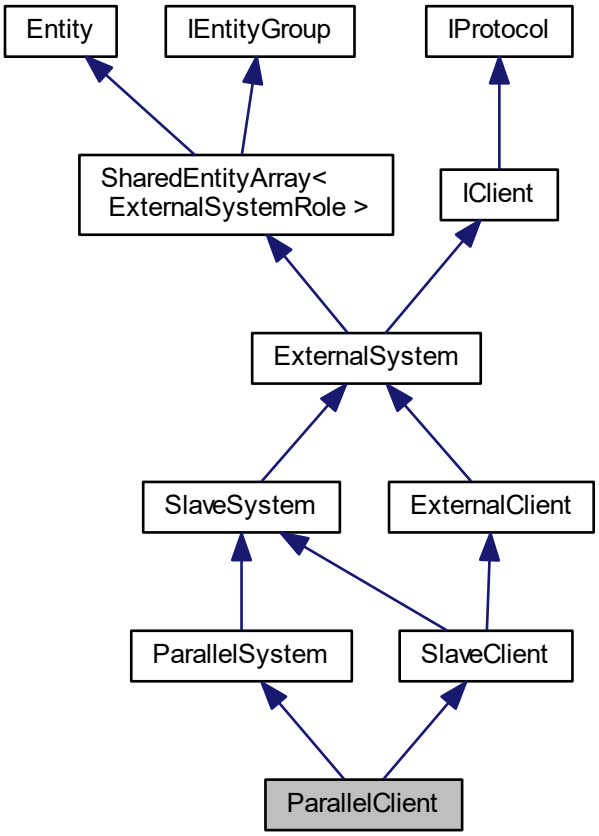
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/Packer.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/Packer.cpp

6.95 ParallelClient Class Reference

A slave parallel client.

```
#include <ParallelClient.hpp>
```

Collaboration diagram for ParallelClient:



Public Member Functions

- ParallelClient ()

  Default Constructor.

Additional Inherited Members

6.95.1 Detailed Description

A slave parallel client.

ParallelClient is a ParallelSystem class specialized in server driver.

[Inherited]

ParallelSystem is a class processing segmented Invoke messages from master.

[Inherited]

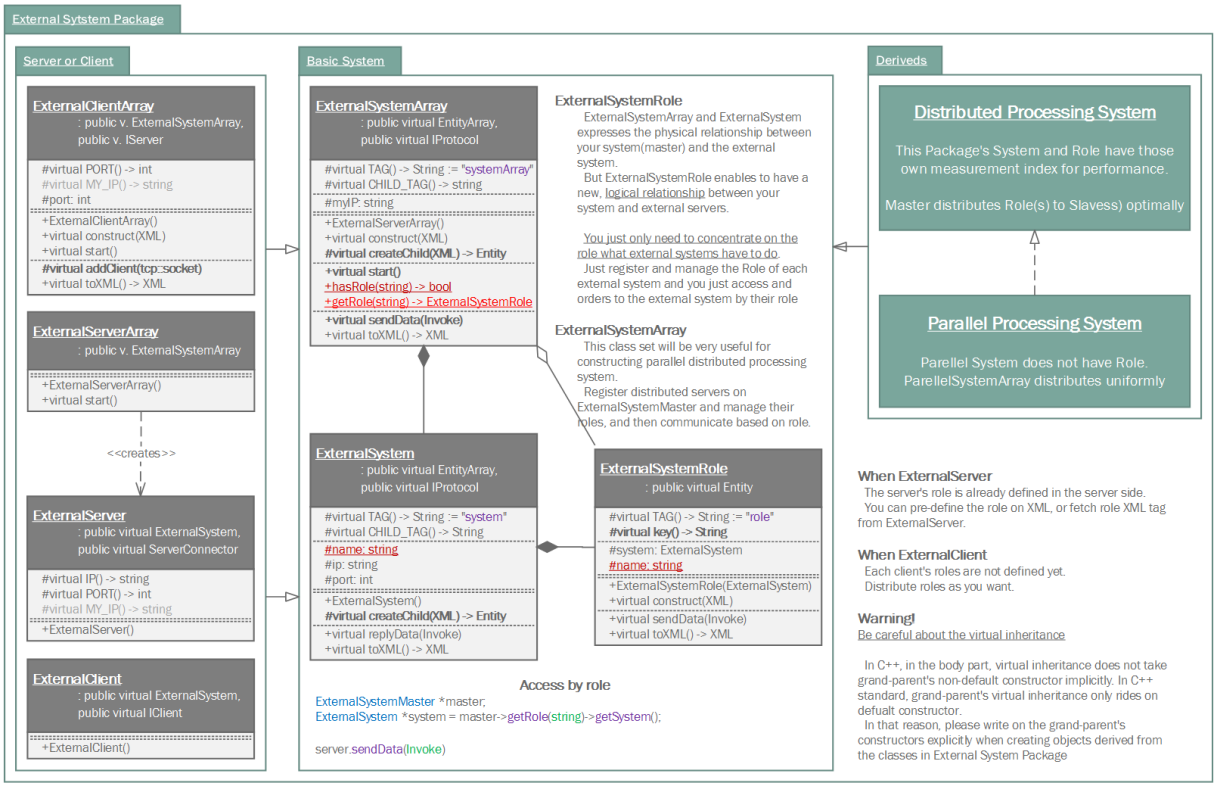
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::Parallel↩ System and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystem↩ Role.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩ SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an External↩ System. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

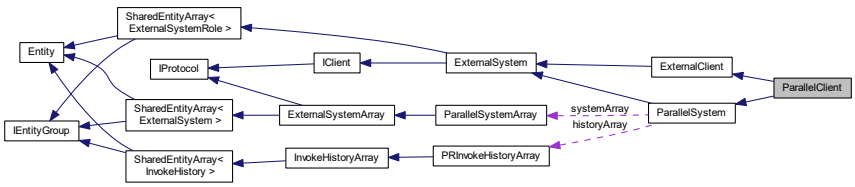
– D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/slave/ParallelClient.hpp

6.96 ParallelClient Class Reference

A network driver for a parallel client.

`#include <ParallelClient.hpp>`

Collaboration diagram for ParallelClient:



Public Member Functions

– ParallelClient ()

Default Constructor.

Additional Inherited Members

6.96.1 Detailed Description

A network driver for a parallel client.

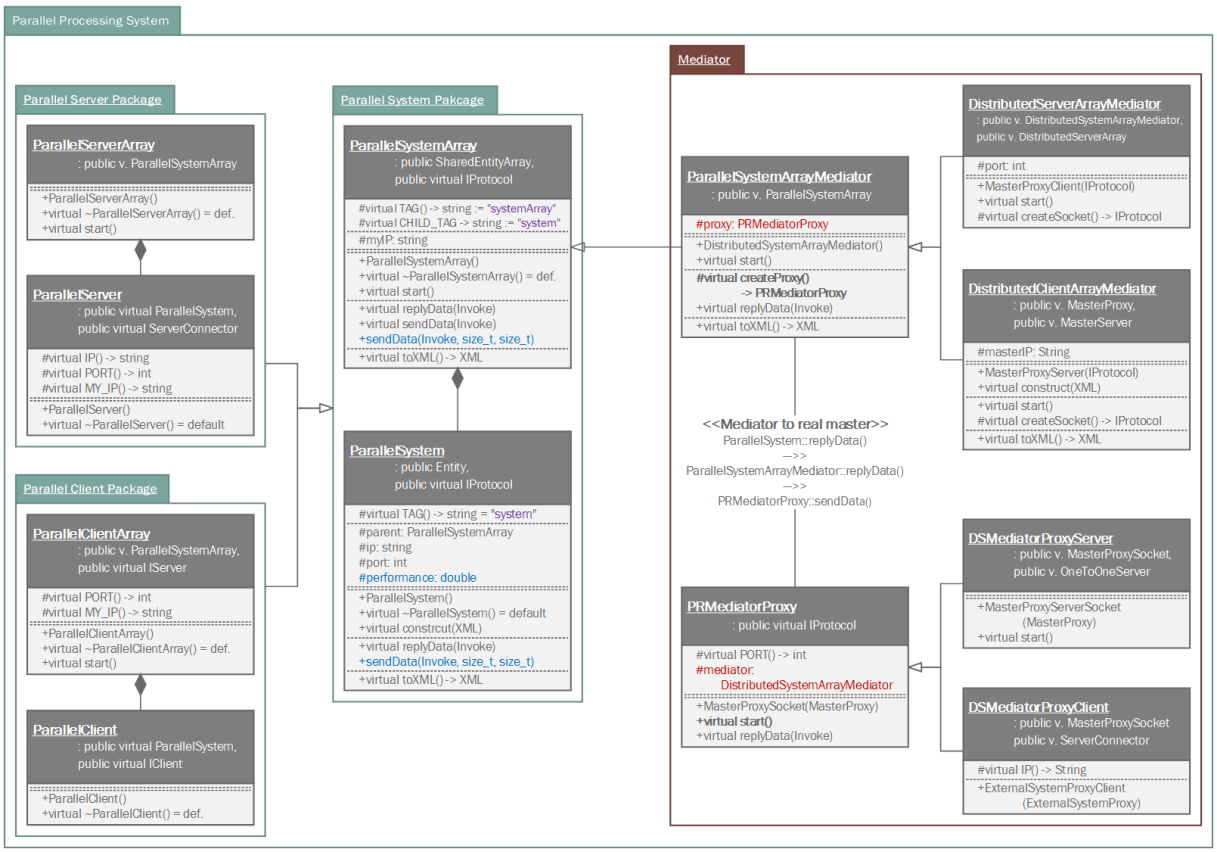
ParallelClient is an ParallelSystem specialized in client driver.

[Inherited]

ParallelSystem class is an ExternalSystem having performance index and history logs of handled Invoke messages with those elapsed times. Those performance index and history logs are used to determine how much segmentation to be allocated.

Each segmentation in requested process (Invoke message) is equivalent. Thus, role of ParallelSystem objects in a ParallelSystemArray are almost same and does not need to specify ExternalSystemRole on ParallelSystem.

- Not a matter to specifying ExternalSystemRole objects to each ParallelSystem. In that case, Invoke messages having segmentation size will be processed by ParallelSystem's own logic and Invoke messages without segmentation size will be handled by ordinary logic of ExternalSystem's own.



[Inherited]

The documentation for this class was generated from the following file:

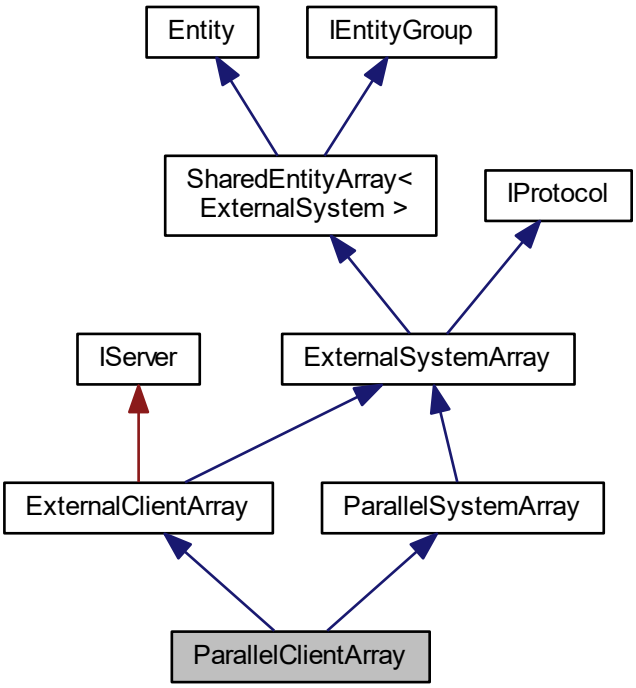
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelClient.hpp

6.97 ParallelClientArray Class Reference

An array of ParallelClient(s).

```
#include <ParallelClientArray.hpp>
```

Collaboration diagram for ParallelClientArray:



Public Member Functions

- ParallelClientArray ()

  Default Constructor.

Additional Inherited Members

6.97.1 Detailed Description

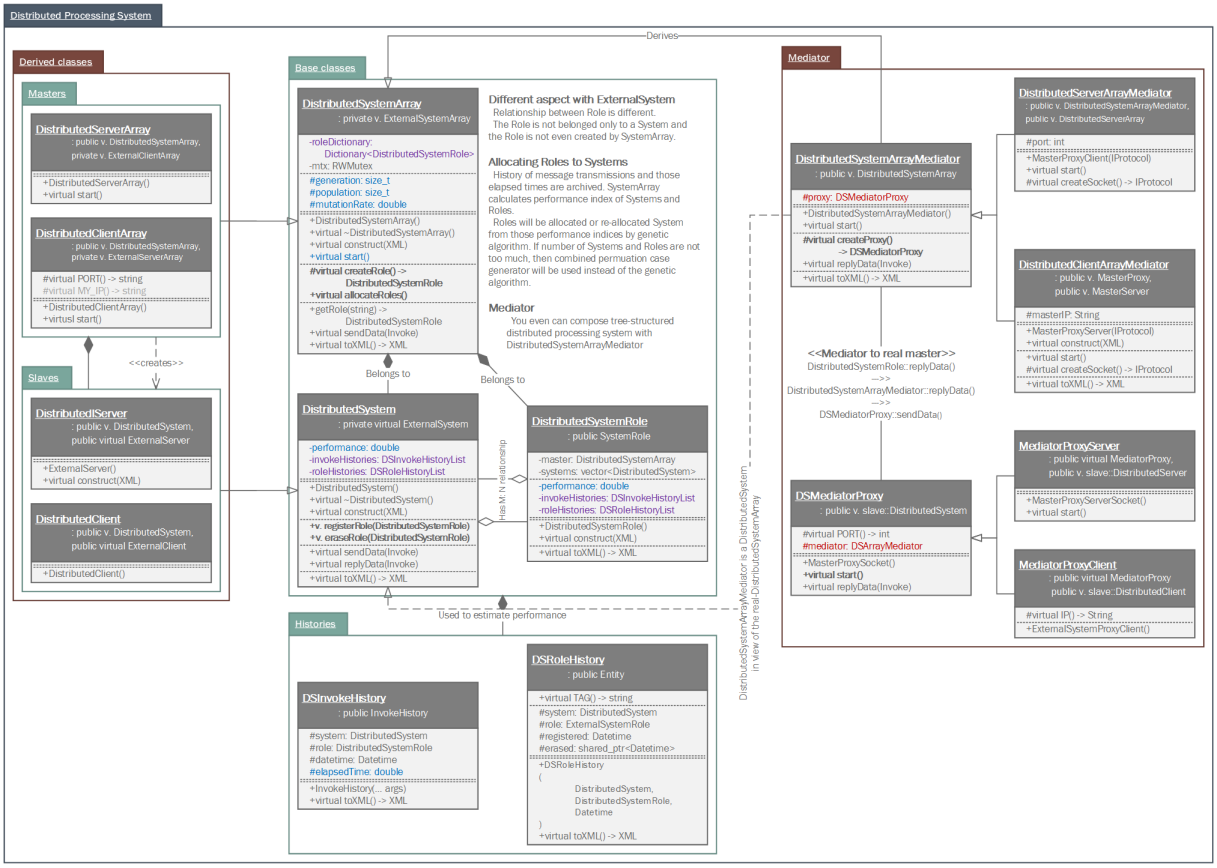
An array of ParallelClient(s).

ParallelClientArray is an ParallelSystemArray specialized in clients' driver.

[Inherited]

DistributedSystemArray class is an ExternalSystemArray and an abstract class containing and managing distributed system drivers within framework of master. The DistributedSystemArray class allocates DistributedSystemRole objects to DistributedSystem objects for optimization.

Unlike the ExternalSystemRole class which is dependent on an ExternalSystem, DistributedSystemRole is not dependent on a DistributedSystem and even can be allocated to multiple DistributedSystem objects. Allocation of DistributedSystemRole(s) to DistributedSystem(s) is determined by estimation of performance index pursuing best distribution. If number of DistributedSystem and DistributedSystemRole objects is not a lot, optimize by combined permuation algorithm. Else the objects are a lot, the allocation will be optimized by genetic algorithm.



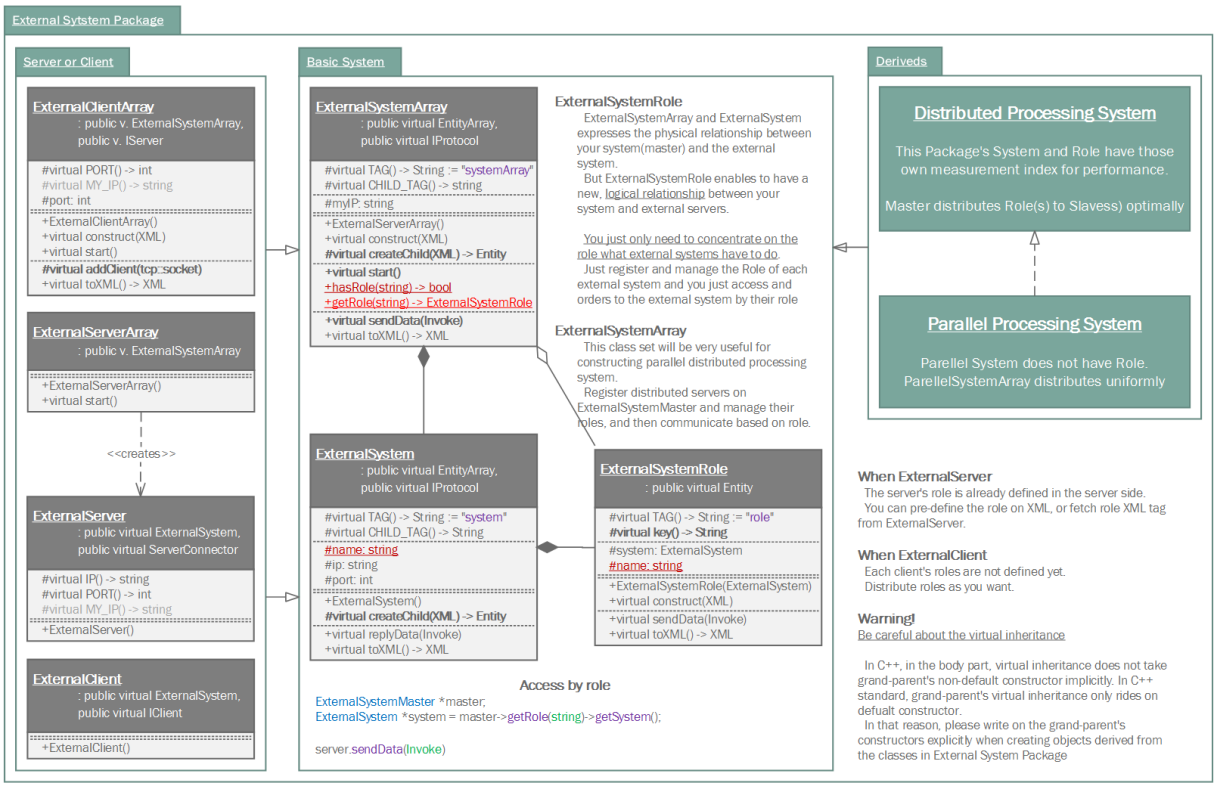
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following files:

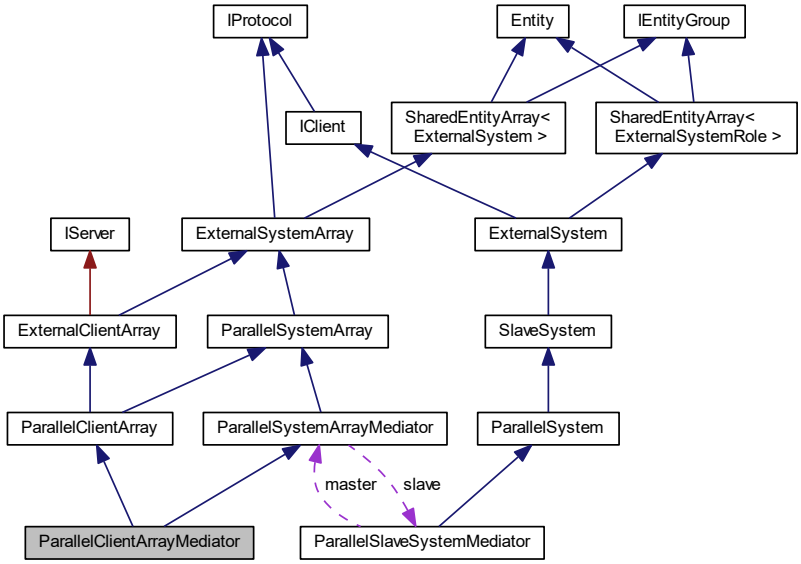
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelClientArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelClientArray.cpp

6.98 ParallelClientArrayMediator Class Reference

A mediator of parallel system between master and slaves.

```
#include <ParallelClientArrayMediator.hpp>
```

Collaboration diagram for ParallelClientArrayMediator:



Public Member Functions

- ParallelClientArrayMediator ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual void start () override

  Start interaction.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Additional Inherited Members

6.98.1 Detailed Description

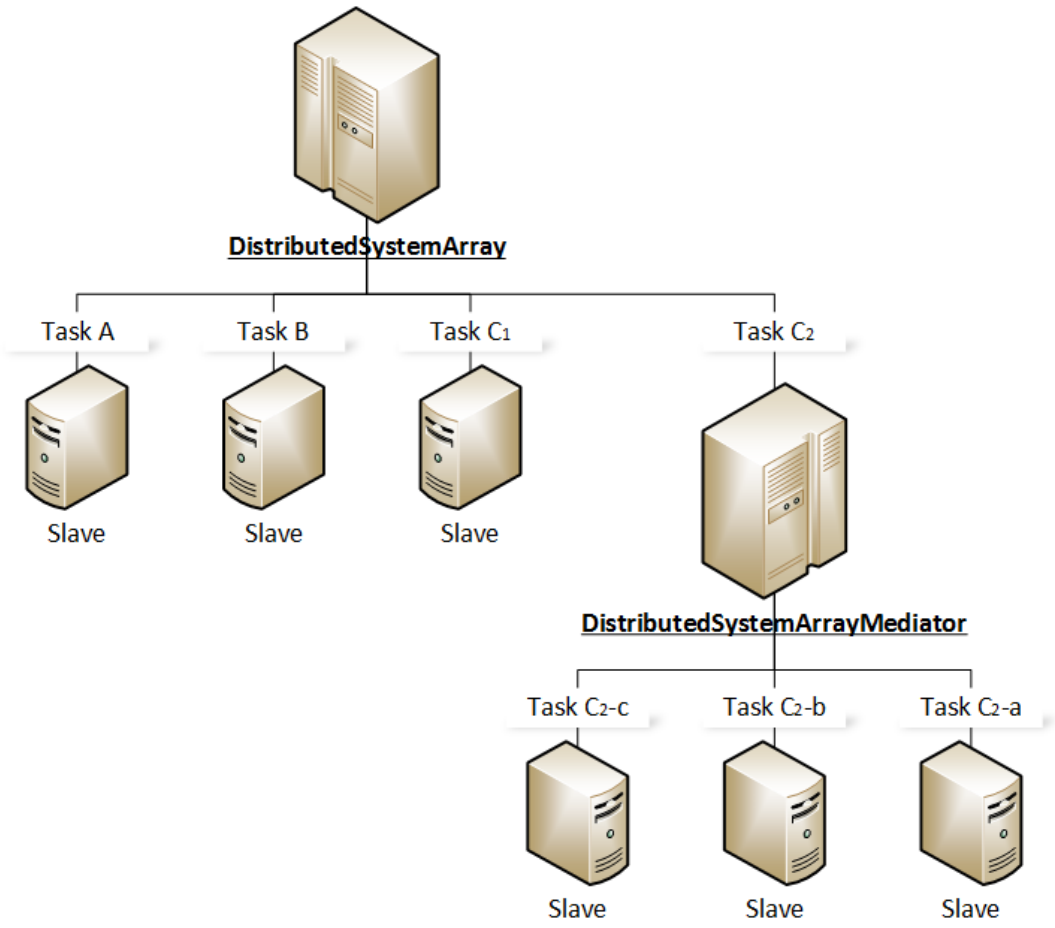
A mediator of parallel system between master and slaves.

ParallelClientArrayMediator is a ParallelSystemArrayMediator specialized in clients' driver. ParallelClient↩ ArrayMediator is a client for its master and also manages children clients' slaves.

[Inherited]

ParallelSystemArrayMediator class is used to realizing a mediator system of tree-structured parallel processing system. ParallelSystemArrayMediator intermediates between origin-master(ParallelSystemArray) and its slaves(slave::ParallelSystem).

ParallelSystemArrayMediator is a master(ParallelSystemArray) within the framework of slaves systems (slave::ParallelSystem objects) and also can be a slave system (slave::ParallelSystem) within framework of its master(ParallelSystemArray) by having ParallelSlaveSystemMediator.



[Inherited]

ParallelSystemArray is an ExternalSystemArray and an abstract class containing. and managing parallel system drvers within framework of master. The ParallelSystemArray class allocates segmented processes (Invoke message containing segmentation size) following by each system's performance index.

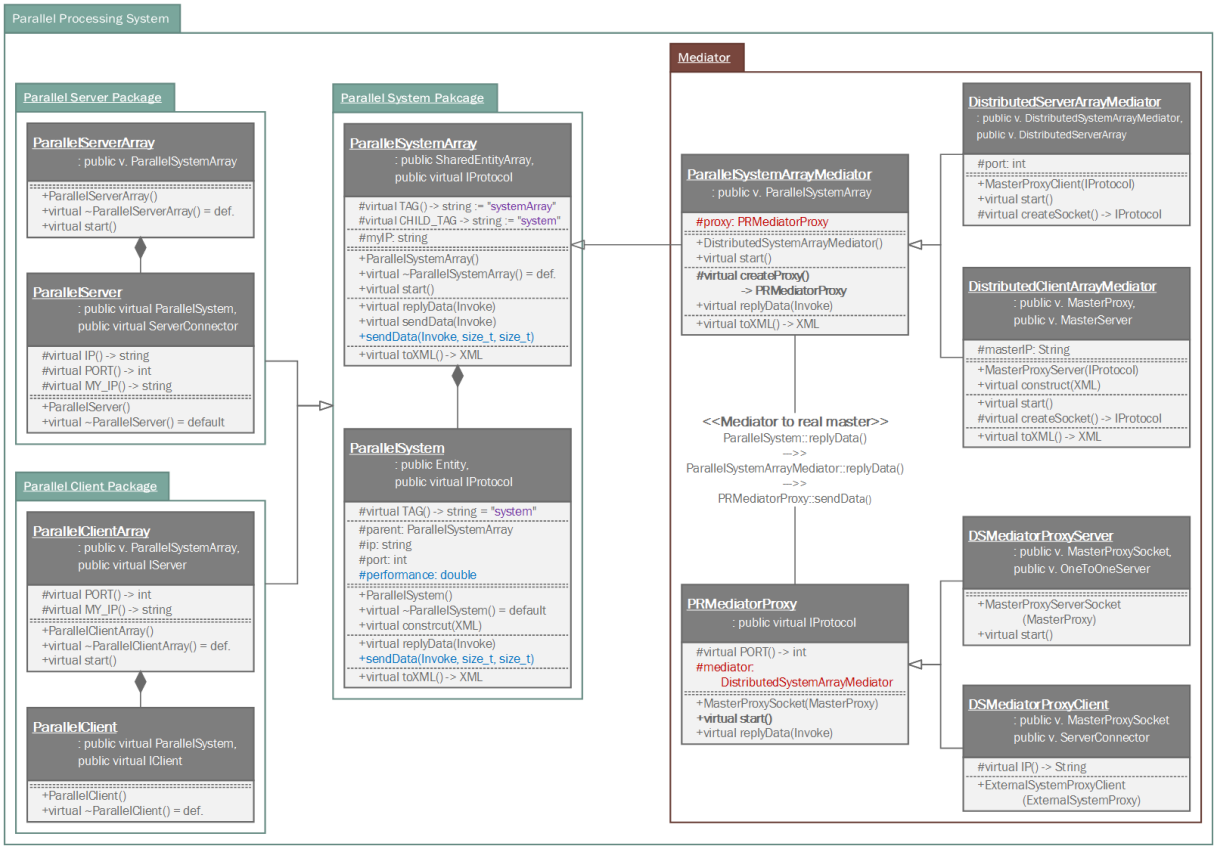
Unlike DistributedSystemArray, ParallelSystemArray does not have such complicate relationships and logics. All segmentation sent by sendData(Invoke, size_t) requires uniform performance. ParallelSystemArray just calculates each system's performance index by elapsing time such easily.

Of course, DistributedSystemArray determines which size of segmentation allocation will be suitable for each system by the performance index

.

Each segmentation in requested process (Invoke message) is equivalent. Thus, role of ParallelSystem objects in a ParallelSystemArray are almost same and does not need to specify ExternalSystemRole on each ParallelSystem.

- Not a matter to specifying ExternalSystemRole objects to each ParallelSystem. In that case, Invoke messages having segmentation size will be processed by ParallelSystemArray's own logic and Invoke messages without segmentation size will be handled by ordinary logic of ExternalSystemArray's own.



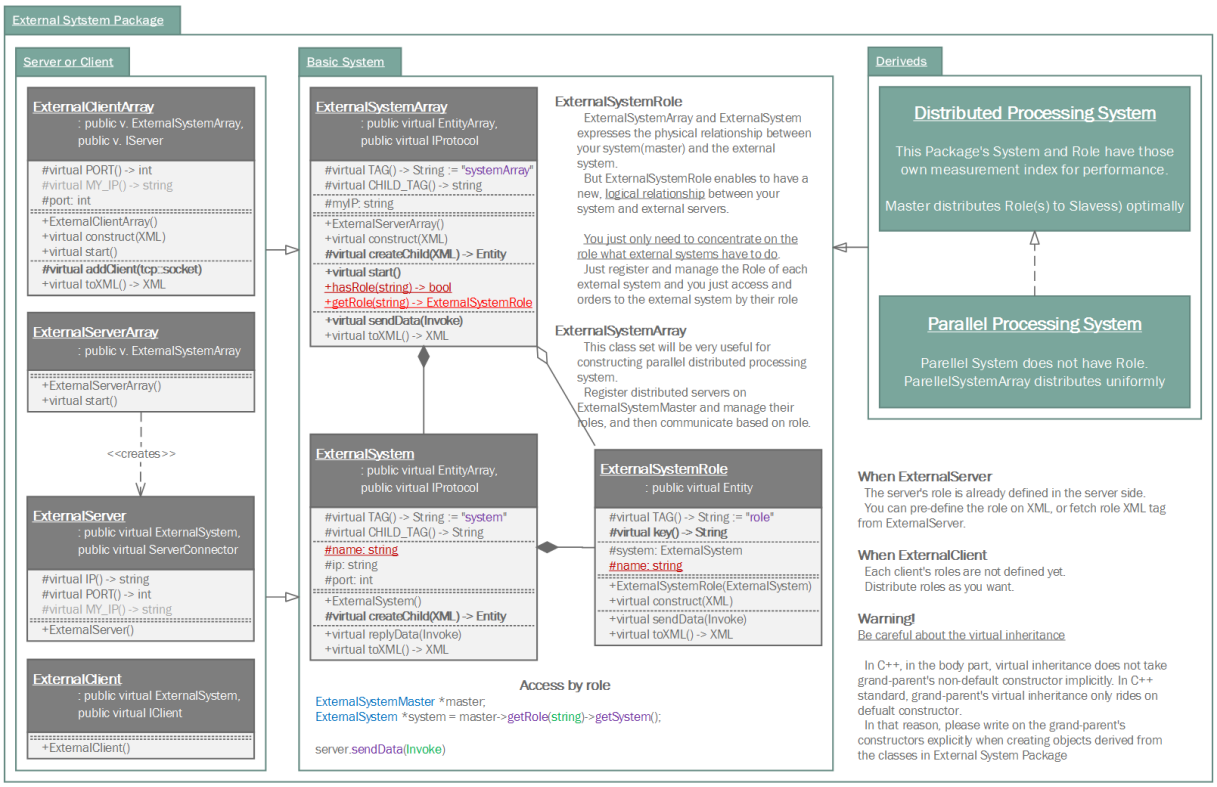
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.98.2 Member Function Documentation

6.98.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from ParallelSystemArrayMediator.

6.98.2.2 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Reimplemented from ParallelSystemArrayMediator.

6.98.2.3 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ParallelSystemArrayMediator.

The documentation for this class was generated from the following file:

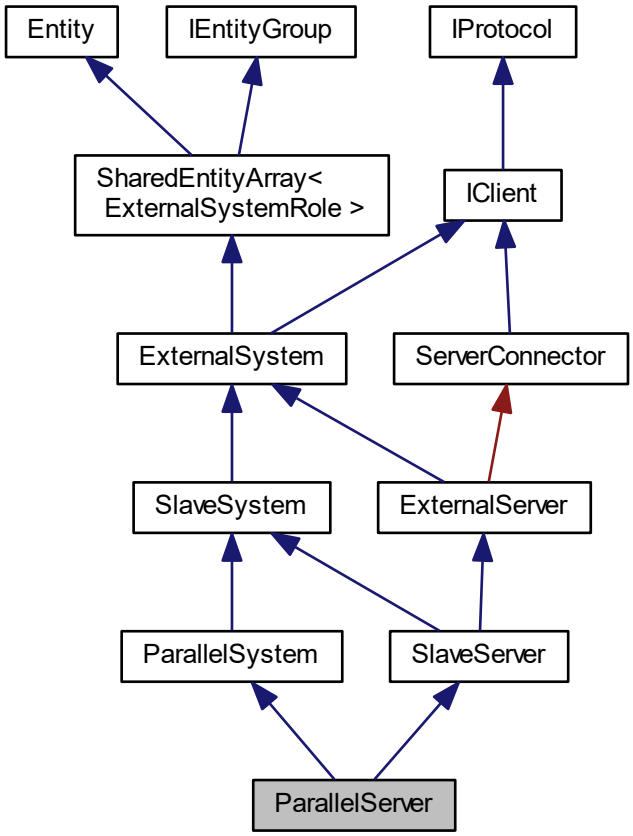
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelClientArrayMediator.↩hpp

6.99 ParallelServer Class Reference

A slave parallel server.

`#include <ParallelServer.hpp>`

Collaboration diagram for ParallelServer:



Public Member Functions

- ParallelServer ()

  Default Constructor.

Additional Inherited Members

6.99.1 Detailed Description

A slave parallel server.

ParallelSystem is a ParallelSystem class specialized in server driver.

[Inherited]

ParallelSystem is a class processing segmented Invoke messages from master.

[Inherited]

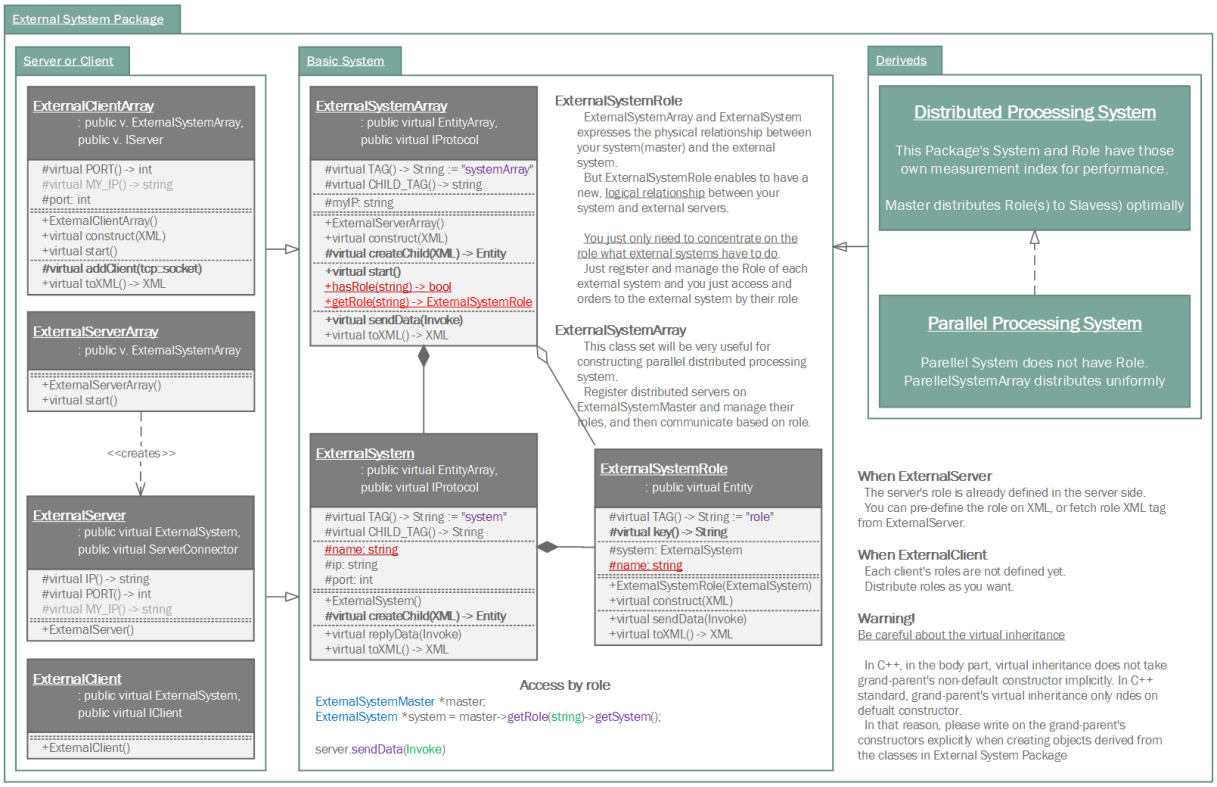
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::Parallel↩System and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystem↩Role.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an External↩System. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

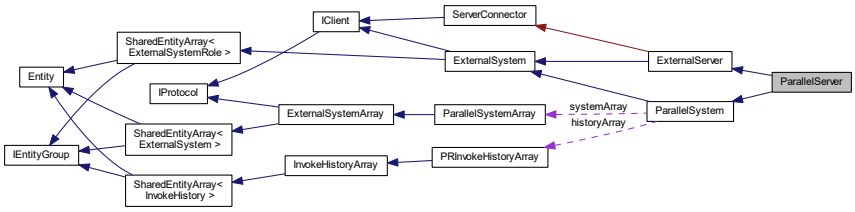
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/slave/ParallelServer.hpp

6.100 ParallelServer Class Reference

A network driver for a parallel server.

```
#include <ParallelServer.hpp>
```

Collaboration diagram for ParallelServer:



Public Member Functions

- ParallelServer ()

  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override

  Construct data of the Entity from an XML object.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override

  Get an XML object represents the EntityGroup.

Additional Inherited Members

6.100.1 Detailed Description

A network driver for a parallel server.

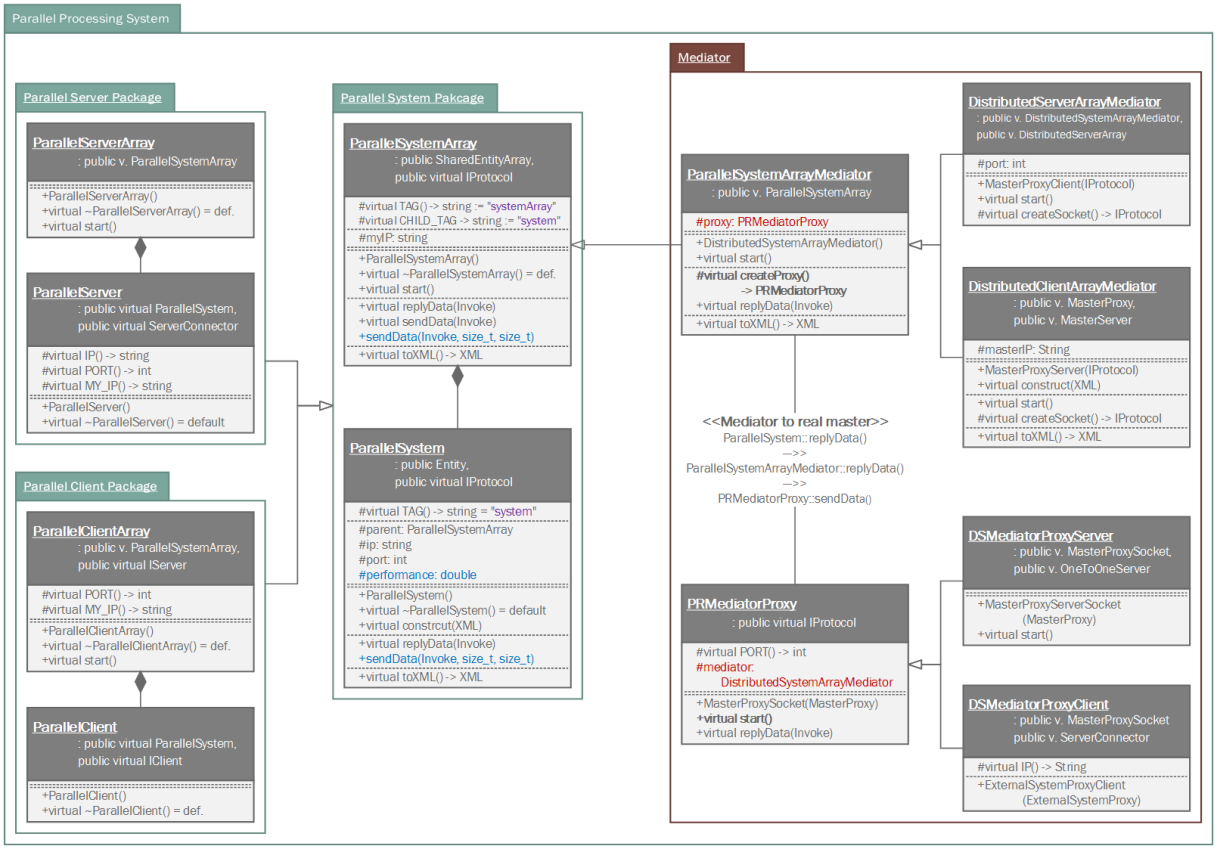
ParallelServer is an ParallelSystem specialized in server driver.

[Inherited]

ParallelSystem class is an ExternalSystem having performance index and history logs of handled Invoke messages with those elapsed times. Those performance index and history logs are used to determine how much segmentation to be allocated.

Each segmentation in requested process (Invoke message) is equivalent. Thus, role of ParallelSystem objects in a ParallelSystemArray are almost same and does not need to specify ExternalSystemRole on Parallel↩System.

– Not a matter to specifying ExternalSystemRole objects to each ParallelSystem. In that case, Invoke messages having segmentation size will be processed by ParallelSystem's own logic and Invoke messages without segmentation size will be handled by ordinary logic of ExternalSystem's own.



[Inherited]

6.100.2 Member Function Documentation

6.100.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from ExternalServer.

6.100.2.2 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override],[virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ExternalServer.

The documentation for this class was generated from the following file:

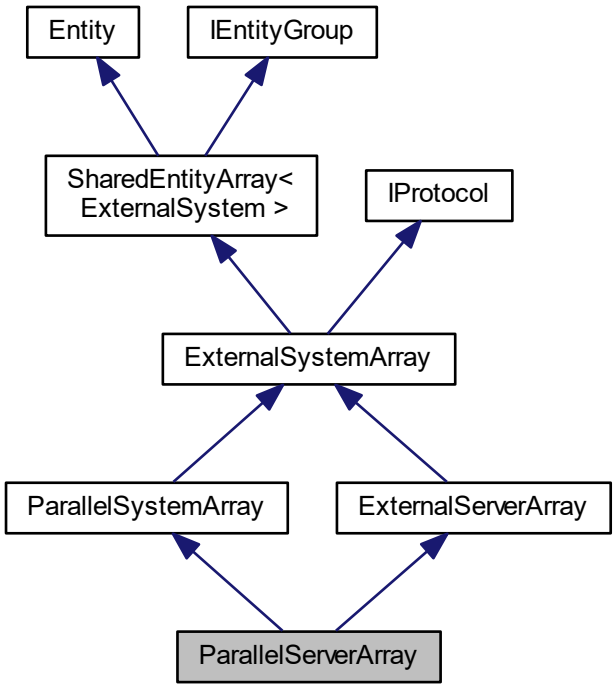
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelServer.hpp

6.101 ParallelServerArray Class Reference

An array of ParallelServer(s).

```
#include <ParallelServerArray.hpp>
```

Collaboration diagram for ParallelServerArray:



Public Member Functions

- ParallelServerArray ()

  Default Constructor.

Additional Inherited Members

6.101.1 Detailed Description

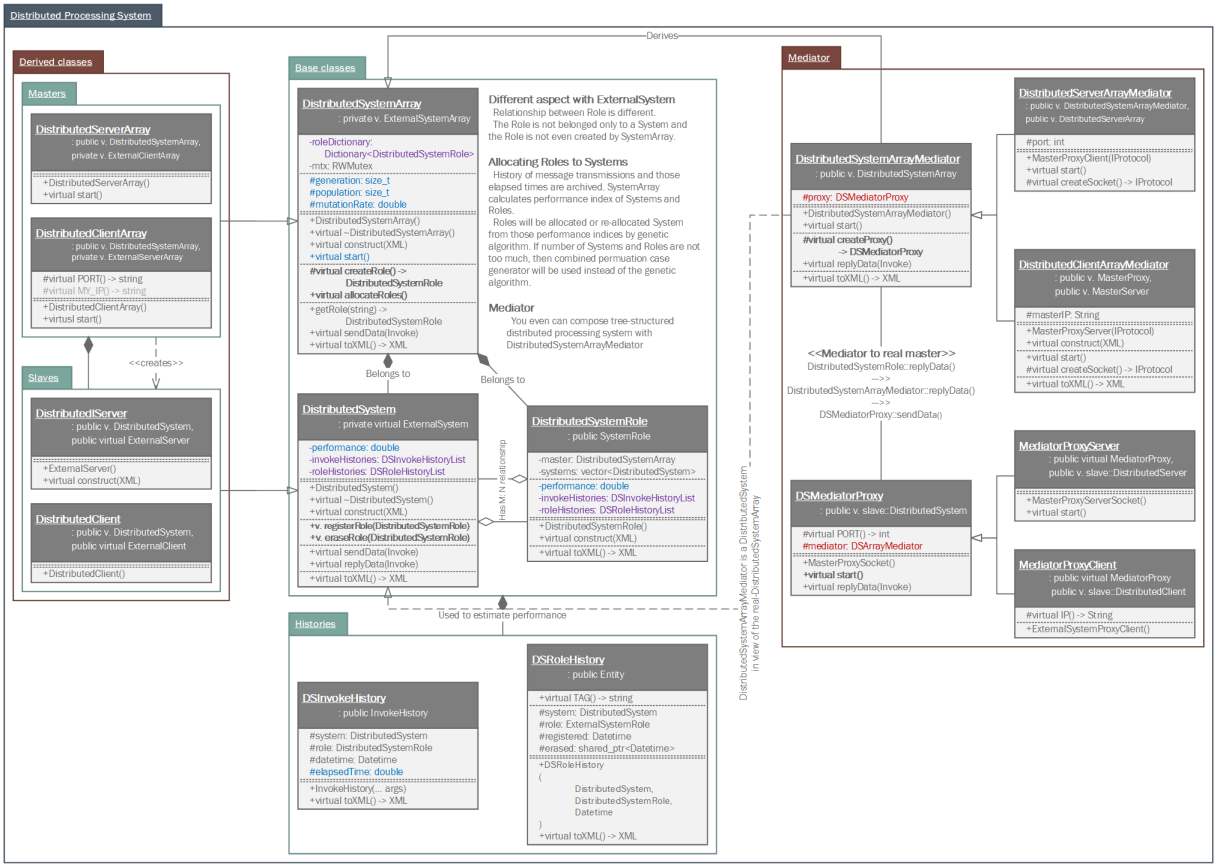
An array of ParallelServer(s).

ParallelServerArray is an ParallelSystemArray specialized in servers' driver.

[Inherited]

DistributedSystemArray class is an ExternalSystemArray and an abstract class containing and managing distributed system drivers within framework of master. The DistributedSystemArray class allocates DistributedSystemRole objects to DistributedSystem objects for optimization.

Unlike the ExternalSystemRole class which is dependent on an ExternalSystem, DistributedSystemRole is not dependent on a DistributedSystem and even can be allocated to multiple DistributedSystem objects. Allocation of DistributedSystemRole(s) to DistributedSystem(s) is determined by estimation of performance index pursuing best distribution. If number of DistributedSystem and DistributedSystemRole objects is not a lot, optimize by combined permuation algorithm. Else the objects are a lot, the allocation will be optimized by genetic algorithm.



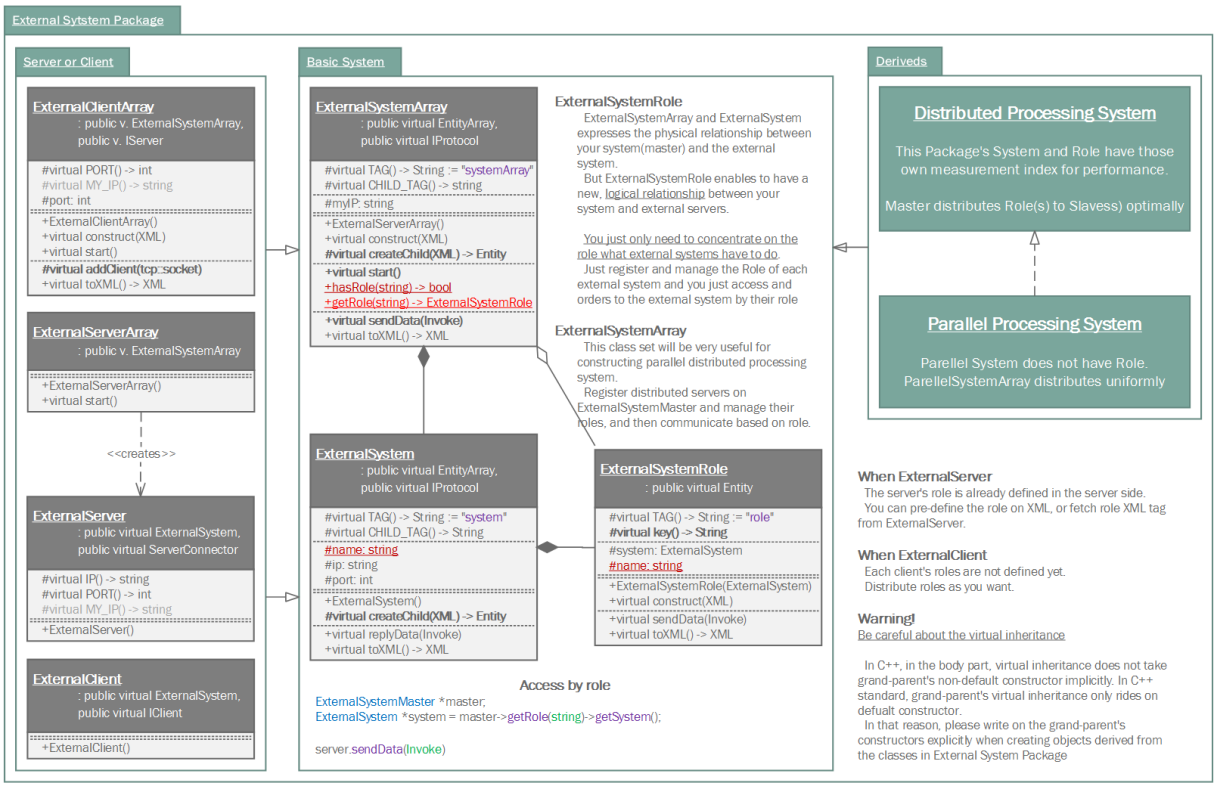
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following files:

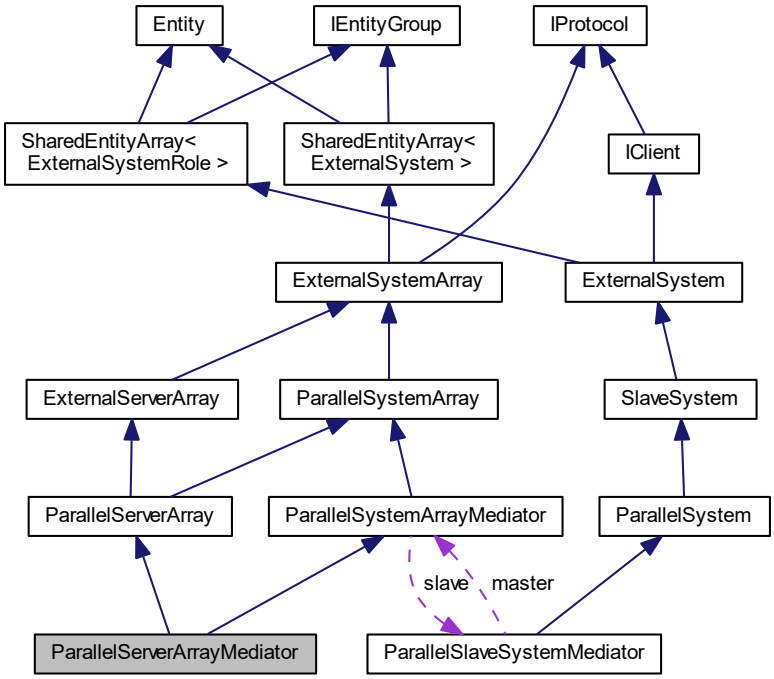
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelServerArray.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelServerArray.cpp

6.102 ParallelServerArrayMediator Class Reference

A mediator of parallel system between master and slaves.

```
#include <ParallelServerArrayMediator.hpp>
```

Collaboration diagram for ParallelServerArrayMediator:



Public Member Functions

- ParallelServerArrayMediator ()

  Default Constructor.
- virtual void start () override

  Start interaction.

Additional Inherited Members

6.102.1 Detailed Description

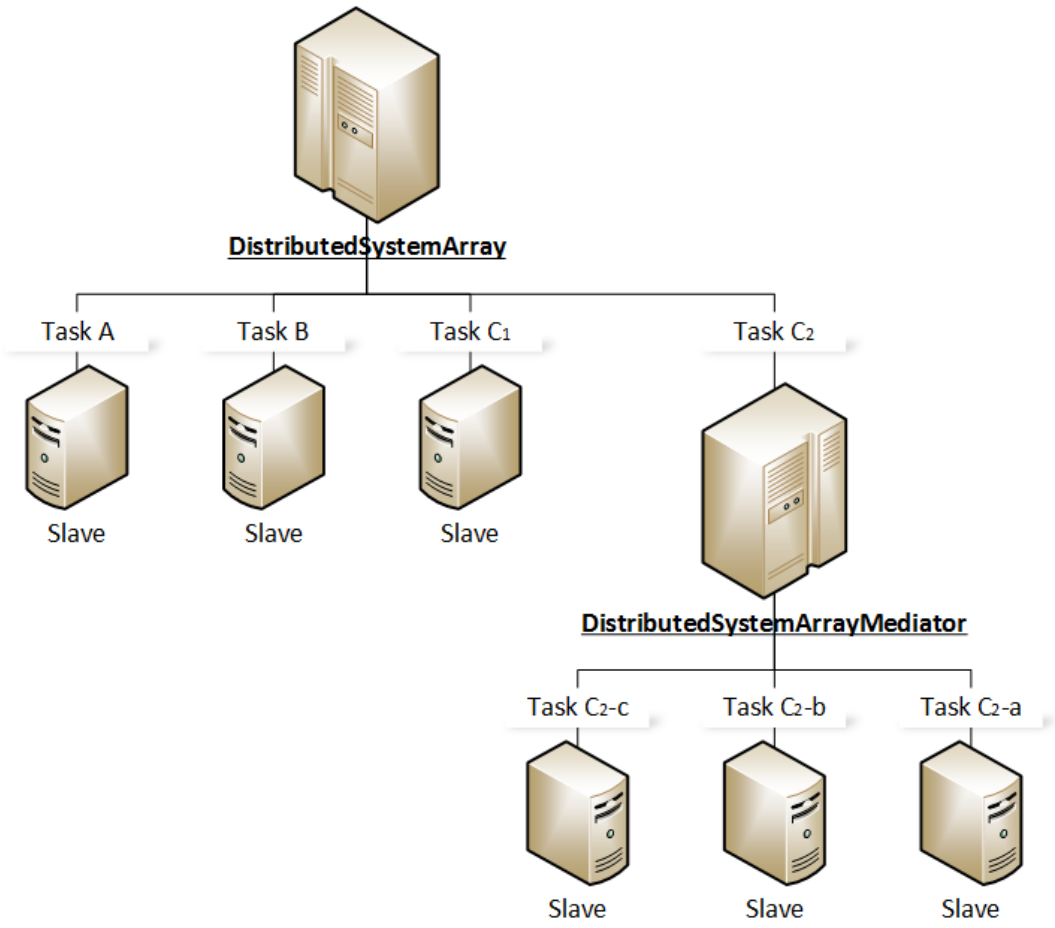
A mediator of parallel system between master and slaves.

ParallelServerArrayMediator is a ParallelSystemArrayMediator specialized in servers' driver. ParallelServer↩ ArrayMediator is a server for its master and also manages children servers' slaves.

[Inherited]

ParallelSystemArrayMediator class is used to realizing a mediator system of tree-structured parallel processing system. ParallelSystemArrayMediator intermediates between origin-master(ParallelSystemArray) and its slaves(slave::ParallelSystem).

ParallelSystemArrayMediator is a master(ParallelSystemArray) within the framework of slaves systems (slave::ParallelSystem objects) and also can be a slave system (slave::ParallelSystem) within framework of its master(ParallelSystemArray) by having ParallelSlaveSystemMediator.



[Inherited]

ParallelSystemArray is an ExternalSystemArray and an abstract class containing. and managing parallel system drvers within framework of master. The ParallelSystemArray class allocates segmented processes (Invoke message containing segmentation size) following by each system's performance index.

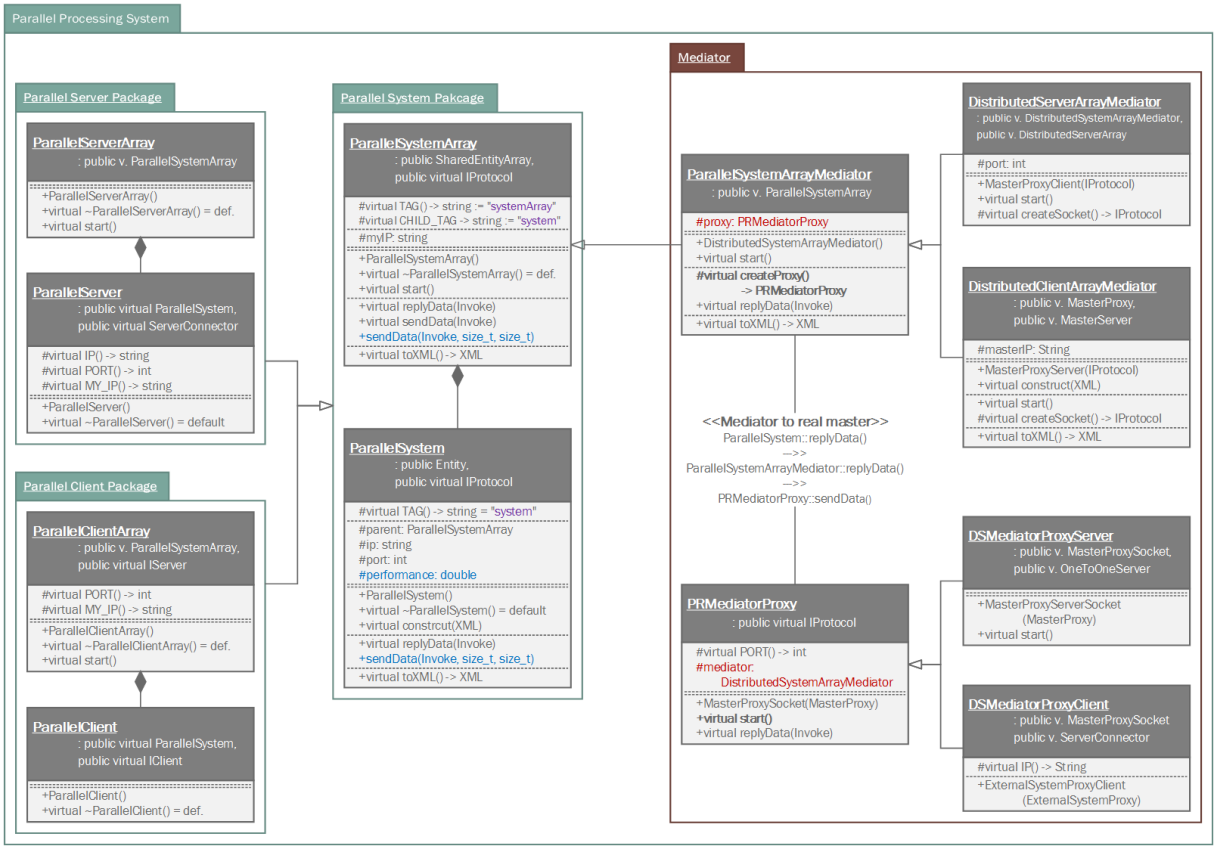
Unlike DistributedSystemArray, ParallelSystemArray does not have such complicate relationships and logics. All segmentation sent by sendData(Invoke, size_t) requires uniform performance. ParallelSystemArray just calculates each system's performance index by elapsing time such easily.

Of course, DistributedSystemArray determines which size of segmentation allocation will be suitable for each system by the performance index

.

Each segmentation in requested process (Invoke message) is equivalent. Thus, role of ParallelSystem objects in a ParallelSystemArray are almost same and does not need to specify ExternalSystemRole on each ParallelSystem.

- Not a matter to specifying ExternalSystemRole objects to each ParallelSystem. In that case, Invoke messages having segmentation size will be processed by ParallelSystemArray's own logic and Invoke messages without segmentation size will be handled by ordinary logic of ExternalSystemArray's own.



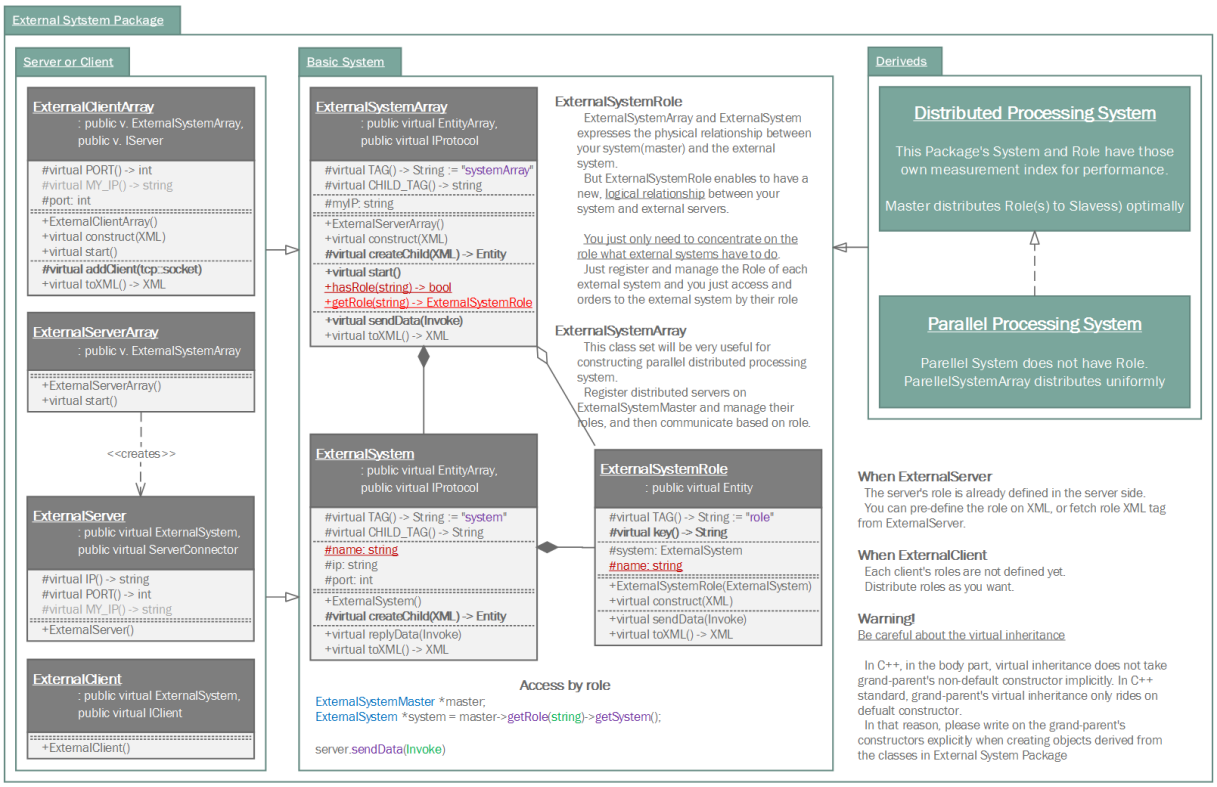
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

- ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.102.2 Member Function Documentation

6.102.2.1 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

- DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Reimplemented from ParallelSystemArrayMediator.

The documentation for this class was generated from the following file:

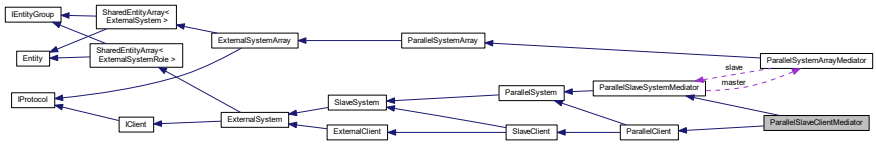
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelServerArrayMediator.↩ hpp

6.103 ParallelSlaveClientMediator Class Reference

A parallel slave client for mediation.

```
#include <ParallelSlaveClientMediator.hpp>
```

Collaboration diagram for ParallelSlaveClientMediator:



Public Member Functions

– ParallelSlaveClientMediator ()

Default Constructor.

Additional Inherited Members

6.103.1 Detailed Description

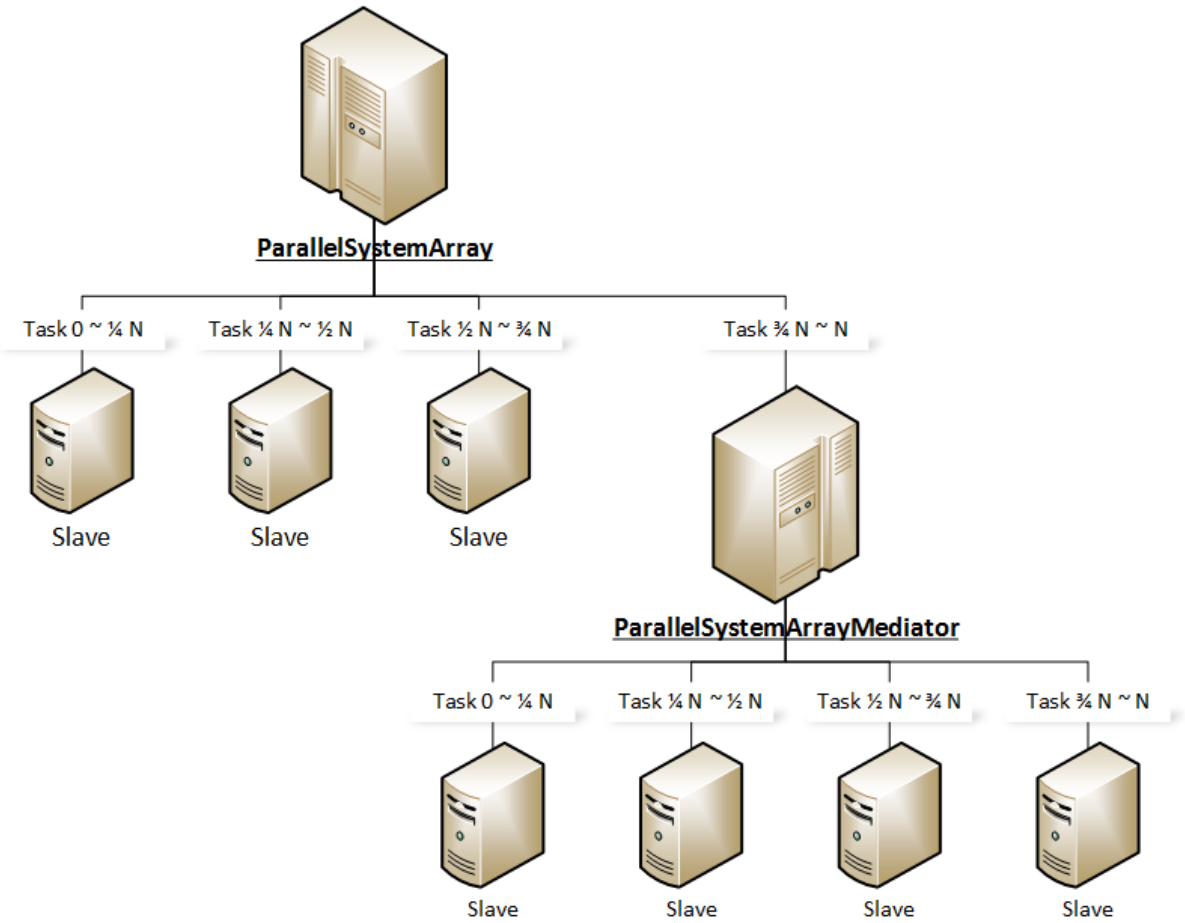
A parallel slave client for mediation.

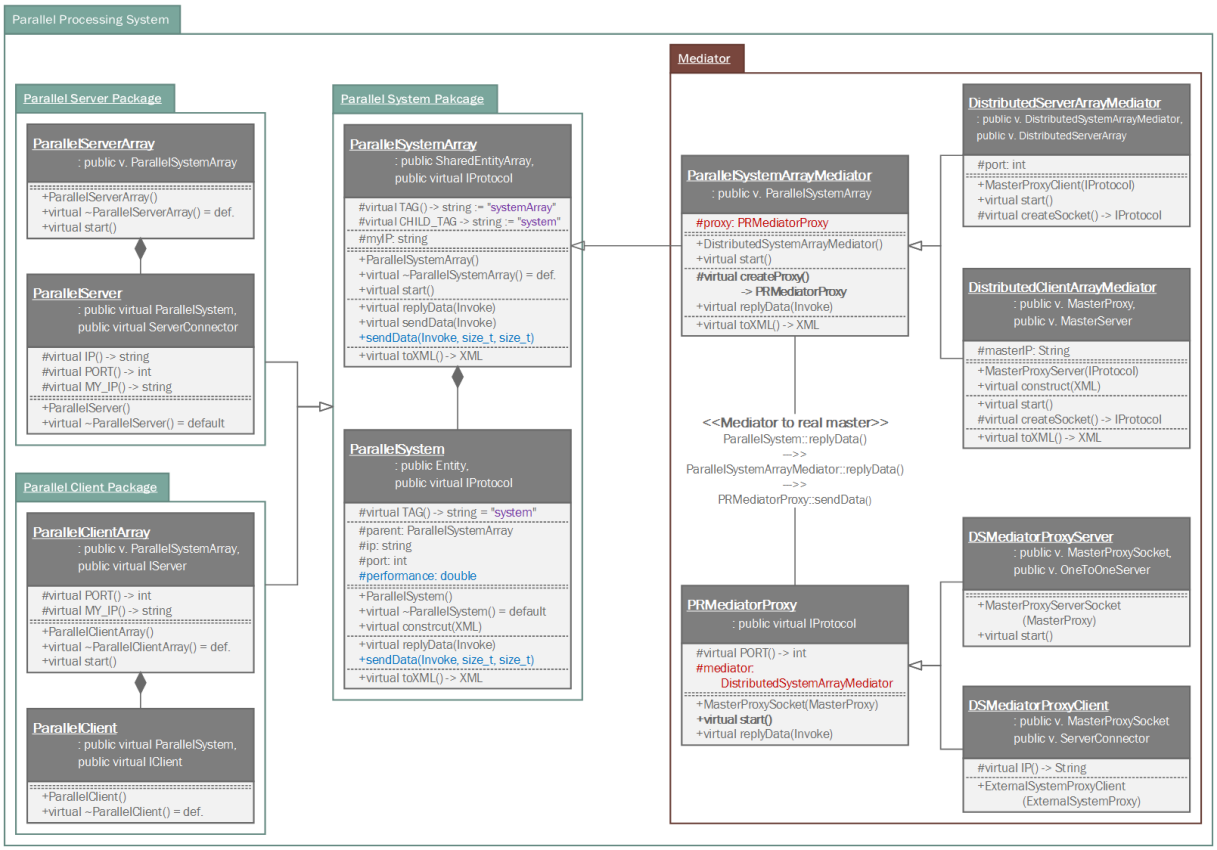
ParallelSlaveClientMediator is a ParallelSlaveSystemMediator class specialized in client driver.

[Inherited]

ParallelSlaveSystemMediator is an abstract class derived from slave::ParallelSystem for realizing tree-structured distriuted processins system. ParallelSlaveSystemMediator acts a slave system's role for its master, on the above level.

ParallelSystemArrayMediator and ParallelSlaveSystemMediator classes are paired to act a mediator system role by acting their own role, master system role for below slaves and slave system role for above master system.


ParallelSystemArray
Task 0 ~ ¼ N
Task ¼ N ~ ½ N
Task ½ N ~ ¾ N
Task ¾ N ~ N
Slave
Slave
Slave
ParallelSystemArrayMediator
Task 0 ~ ¼ N
Task ¼ N ~ ½ N
Task ½ N ~ ¾ N
Task ¾ N ~ N
Slave
Slave
Slave
Slave


Parallel Processing System
Parallel Server Package
ParallelServerArray
: public v. ParallelSystemArray
+ParallelServerArray()
+virtual ~ParallelServerArray() = def.
+virtual start()
ParallelServer
: public virtual ParallelSystem,
public virtual ServerConnector
#virtual IP() -> string
#virtual PORT() -> int
#virtual MY_IP() -> string
+ParallelServer()
+virtual ~ParallelServer() = default
Parallel Client Package
ParallelClientArray
: public v. ParallelSystemArray,
public virtual IServer
#virtual PORT() -> int
#virtual MY_IP() -> string
+ParallelClientArray()
+virtual ~ParallelClientArray() = def.
+virtual start()
ParallelClient
: public virtual ParallelSystem,
public virtual IClient
+ParallelClient()
+virtual ~ParallelClient() = def.
Parallel System Pakcage
ParallelSystemArray
: public SharedEntityArray,
public virtual IProtocol
#virtual TAG() -> string := "systemArray"
#virtual CHILD_TAG -> string := "system"
#myIP: string
+ParallelSystemArray()
+virtual ~ParallelSystemArray() = def.
+virtual start()
+virtual replyData(Invoke)
+virtual sendData(Invoke)
+sendData(Invoke, size_t, size_t)
+virtual toXML() -> XML
ParallelSystem
: public Entity,
public virtual IProtocol
#virtual TAG() -> string = "system"
#parent: ParallelSystemArray
#ip: string
#port: int
#performance: double
+ParallelSystem()
+virtual ~ParallelSystem() = default
+virtual constrcut(XML)
+virtual replyData(Invoke)
+sendData(Invoke, size_t, size_t)
+virtual toXML() -> XML
Mediator
ParallelSystemArrayMediator
: public v. ParallelSystemArray
#proxy: PRMediatorProxy
+DistributedSystemArrayMediator()
+virtual start()
#virtual createProxy()
-> PRMediatorProxy
+virtual replyData(Invoke)
+virtual toXML() -> XML
DistributedServerArrayMediator
: public v. DistributedSystemArrayMediator,
public v. DistributedServerArray
#port: int
+MasterProxyClient(IProtocol)
+virtual start()
#virtual createSocket() -> IProtocol
DistributedClientArrayMediator
: public v. MasterProxy,
public v. MasterServer
#masterIP: String
+MasterProxyServer(IProtocol)
+virtual construct(XML)
+virtual start()
#virtual createSocket() -> IProtocol
+virtual toXML() -> XML
<<Mediator to real master>>
ParallelSystem::replyData()
-->>
ParallelSystemArrayMediator::replyData()
-->>
PRMediatorProxy::sendData()
PRMediatorProxy
: public virtual IProtocol
#virtual PORT() -> int
#mediator:
DistributedSystemArrayMediator
+MasterProxySocket(MasterProxy)
+virtual start()
+virtual replyData(Invoke)
DSMediatorProxyServer
: public v. MasterProxySocket,
public v. OneToOneServer
+MasterProxyServerSocket
(MasterProxy)
+virtual start()
DSMediatorProxyClient
: public v. MasterProxySocket
public v. ServerConnector
#virtual IP() -> String
+ExternalSystemProxyClient
(ExternalSystemProxy)

[Inherited]

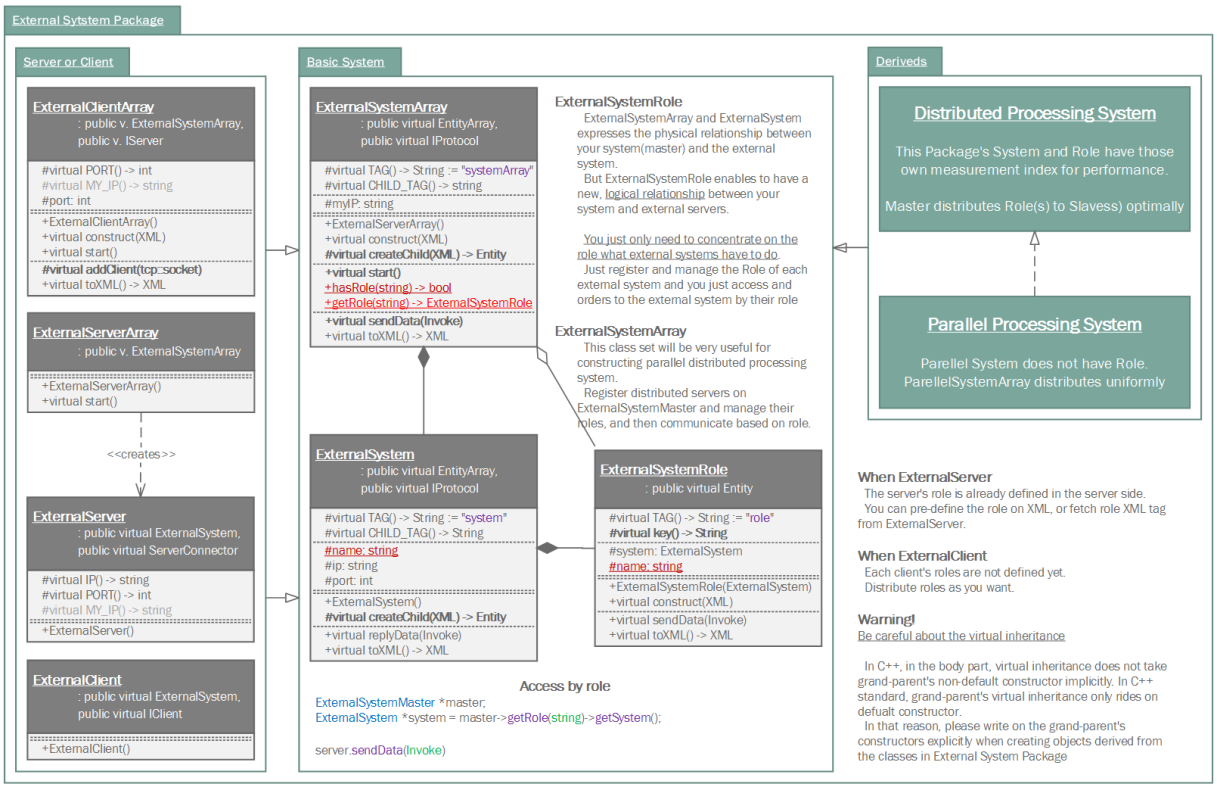
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::Parallel↩System and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystem↩Role.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an External↩System. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

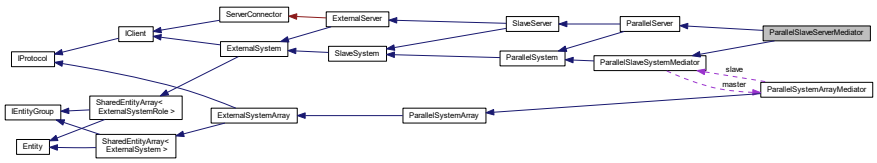
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/slave/ParallelSlaveClientMediator.hpp

6.104 ParallelSlaveServerMediator Class Reference

A parallel slave server for mediation.

```
#include <ParallelSlaveServerMediator.hpp>
```

Collaboration diagram for ParallelSlaveServerMediator:



Public Member Functions

- ParallelSlaveServerMediator ()

  Default Constructor.

Additional Inherited Members

6.104.1 Detailed Description

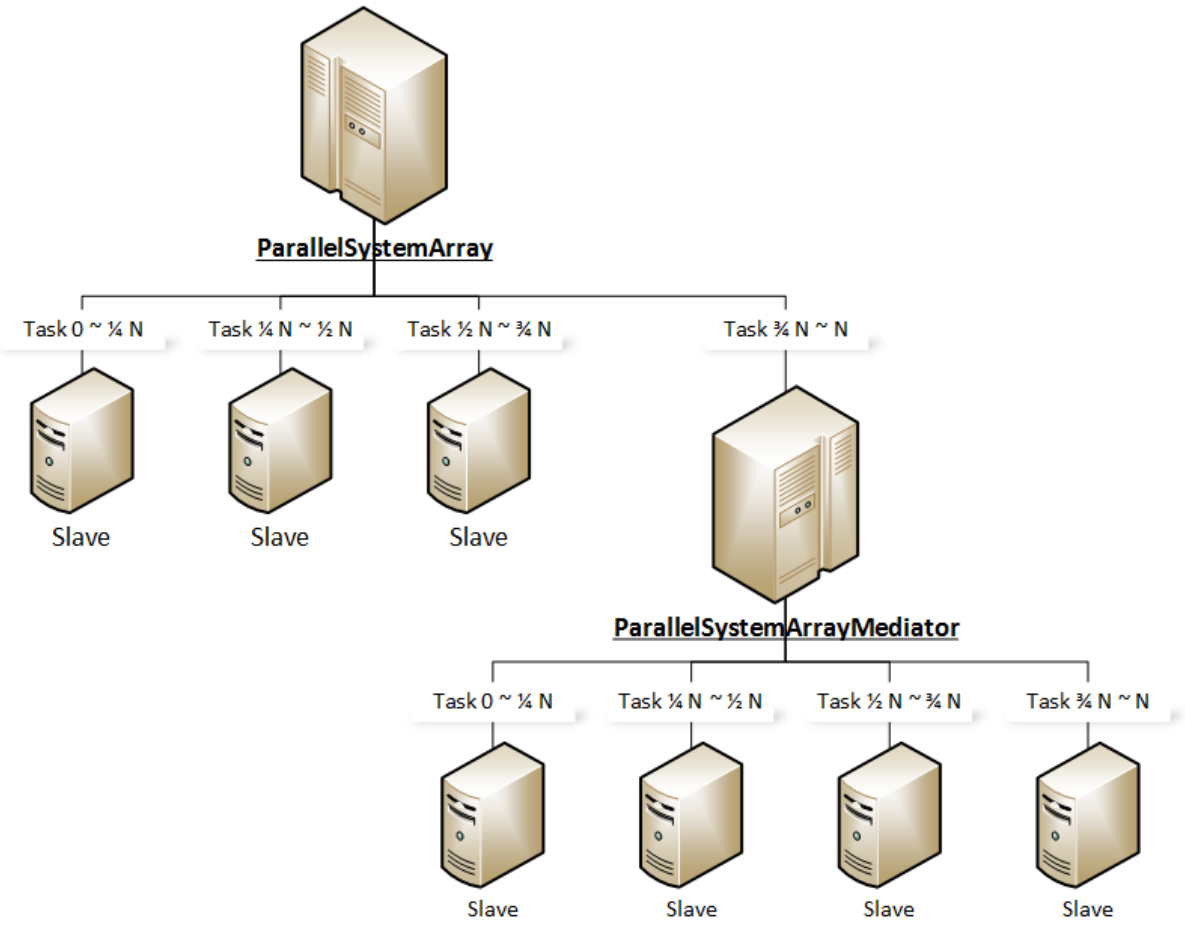
A parallel slave server for mediation.

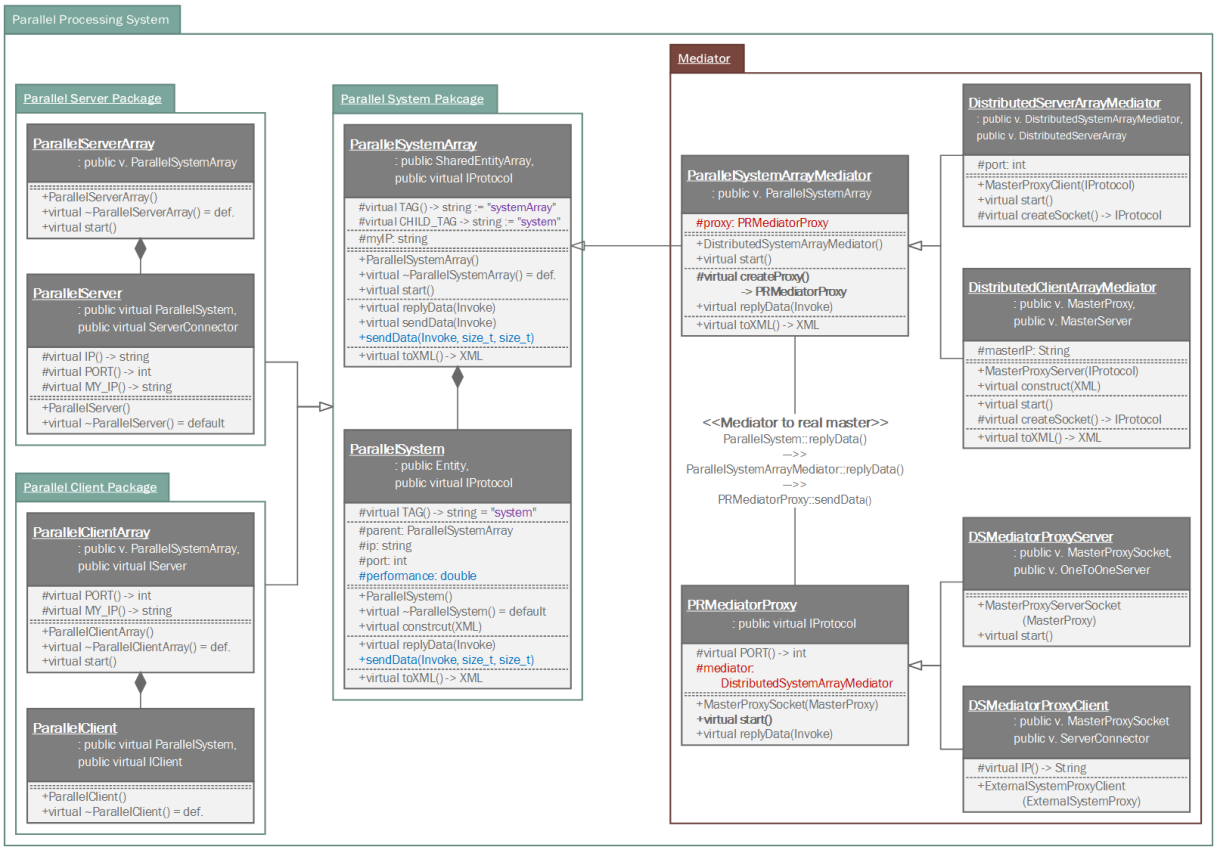
ParallelSlaveServerMediator is a ParallelSlaveSystemMediator class specialized in server driver.

[Inherited]

ParallelSlaveSystemMediator is an abstract class derived from slave::ParallelSystem for realizing tree-structured distriuted processins system. ParallelSlaveSystemMediator acts a slave system's role for its master, on the above level.

ParallelSystemArrayMediator and ParallelSlaveSystemMediator classes are paired to act a mediator system role by acting their own role, master system role for below slaves and slave system role for above master system.


ParallelSystemArray
Task 0 ~ ¼ N
Task ¼ N ~ ½ N
Task ½ N ~ ¾ N
Task ¾ N ~ N
Slave
Slave
Slave
ParallelSystemArrayMediator
Task 0 ~ ¼ N
Task ¼ N ~ ½ N
Task ½ N ~ ¾ N
Task ¾ N ~ N
Slave
Slave
Slave
Slave


Parallel Processing System
Parallel Server Package
ParallelServerArray
: public v. ParallelSystemArray
+ParallelServerArray()
+virtual ~ParallelServerArray() = def.
+virtual start()
ParallelServer
: public virtual ParallelSystem,
public virtual ServerConnector
#virtual IP() -> string
#virtual PORT() -> int
#virtual MY_IP() -> string
+ParallelServer()
+virtual ~ParallelServer() = default
Parallel Client Package
ParallelClientArray
: public v. ParallelSystemArray,
public virtual IServer
#virtual PORT() -> int
#virtual MY_IP() -> string
+ParallelClientArray()
+virtual ~ParallelClientArray() = def.
+virtual start()
ParallelClient
: public virtual ParallelSystem,
public virtual IClient
+ParallelClient()
+virtual ~ParallelClient() = def.
Parallel System Pakcage
ParallelSystemArray
: public SharedEntityArray,
public virtual IProtocol
#virtual TAG() -> string := "systemArray"
#virtual CHILD_TAG -> string := "system"
#myIP: string
+ParallelSystemArray()
+virtual ~ParallelSystemArray() = def.
+virtual start()
+virtual replyData(Invoke)
+virtual sendData(Invoke)
+sendData(Invoke, size_t, size_t)
+virtual toXML() -> XML
ParallelSystem
: public Entity,
public virtual IProtocol
#virtual TAG() -> string = "system"
#parent: ParallelSystemArray
#ip: string
#port: int
#performance: double
+ParallelSystem()
+virtual ~ParallelSystem() = default
+virtual constrcut(XML)
+virtual replyData(Invoke)
+sendData(Invoke, size_t, size_t)
+virtual toXML() -> XML
Mediator
ParallelSystemArrayMediator
: public v. ParallelSystemArray
#proxy: PRMediatorProxy
+DistributedSystemArrayMediator()
+virtual start()
#virtual createProxy()
-> PRMediatorProxy
+virtual replyData(Invoke)
+virtual toXML() -> XML
DistributedServerArrayMediator
: public v. DistributedSystemArrayMediator,
public v. DistributedServerArray
#port: int
+MasterProxyClient(IProtocol)
+virtual start()
#virtual createSocket() -> IProtocol
DistributedClientArrayMediator
: public v. MasterProxy,
public v. MasterServer
#masterIP: String
+MasterProxyServer(IProtocol)
+virtual construct(XML)
+virtual start()
#virtual createSocket() -> IProtocol
+virtual toXML() -> XML
<<Mediator to real master>>
ParallelSystem::replyData()
-->>
ParallelSystemArrayMediator::replyData()
-->>
PRMediatorProxy::sendData()
PRMediatorProxy
: public virtual IProtocol
#virtual PORT() -> int
#mediator:
DistributedSystemArrayMediator
+MasterProxySocket(MasterProxy)
+virtual start()
+virtual replyData(Invoke)
DSMediatorProxyServer
: public v. MasterProxySocket,
public v. OneToOneServer
+MasterProxyServerSocket
(MasterProxy)
+virtual start()
DSMediatorProxyClient
: public v. MasterProxySocket
public v. ServerConnector
#virtual IP() -> String
+ExternalSystemProxyClient
(ExternalSystemProxy)

[Inherited]

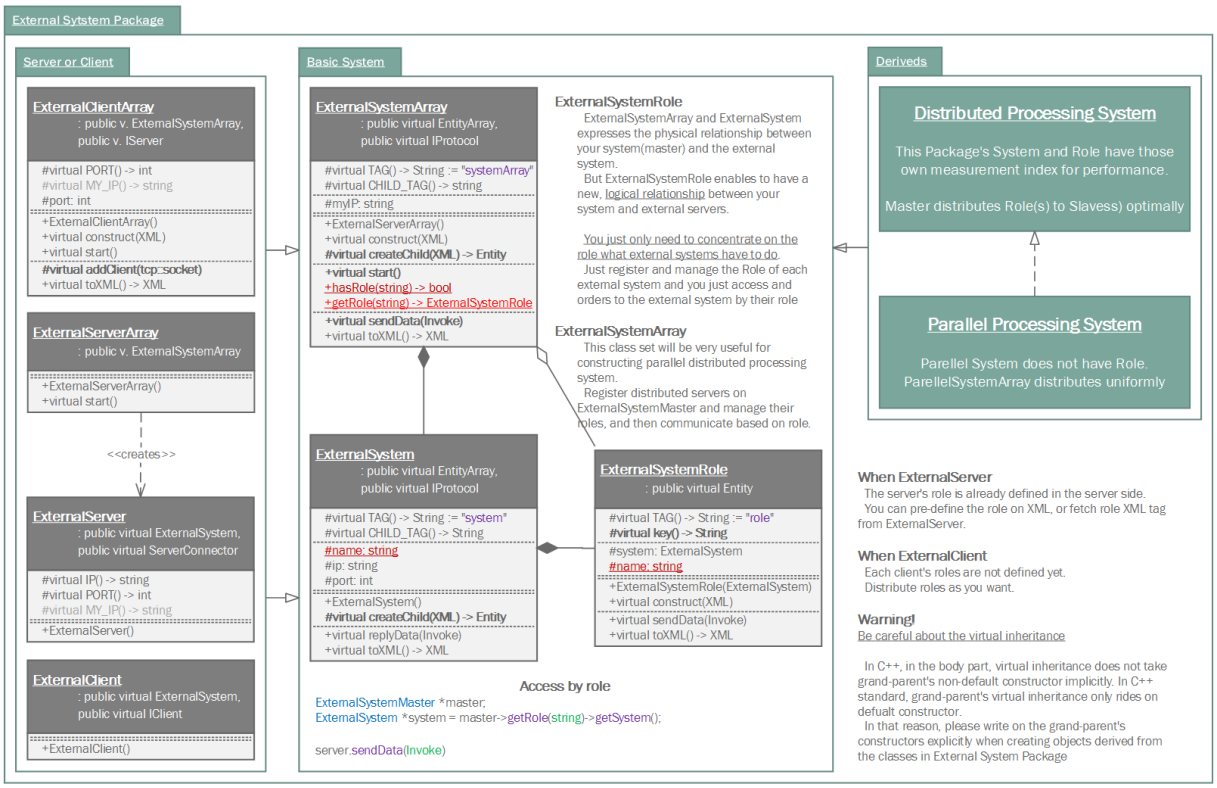
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::Parallel↩System and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystem↩Role.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an External↩System. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

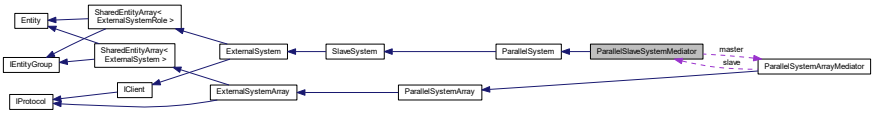
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/slave/ParallelSlaveServerMediator.↩ hpp

6.105 ParallelSlaveSystemMediator Class Reference

A parallel slave system for mediation.

`#include <ParallelSlaveSystemMediator.hpp>`

Collaboration diagram for ParallelSlaveSystemMediator:



Public Member Functions

- ParallelSlaveSystemMediator ()

  Default Constructor.
- virtual void replyData (std::shared_ptr< Invoke >) override

  Handling replied message from an external system.
- virtual auto TAG () const -> std::string override

  A tag name when represented by XML.

Protected Attributes

- ParallelSystemArrayMediator * master

  A master system for mediation.

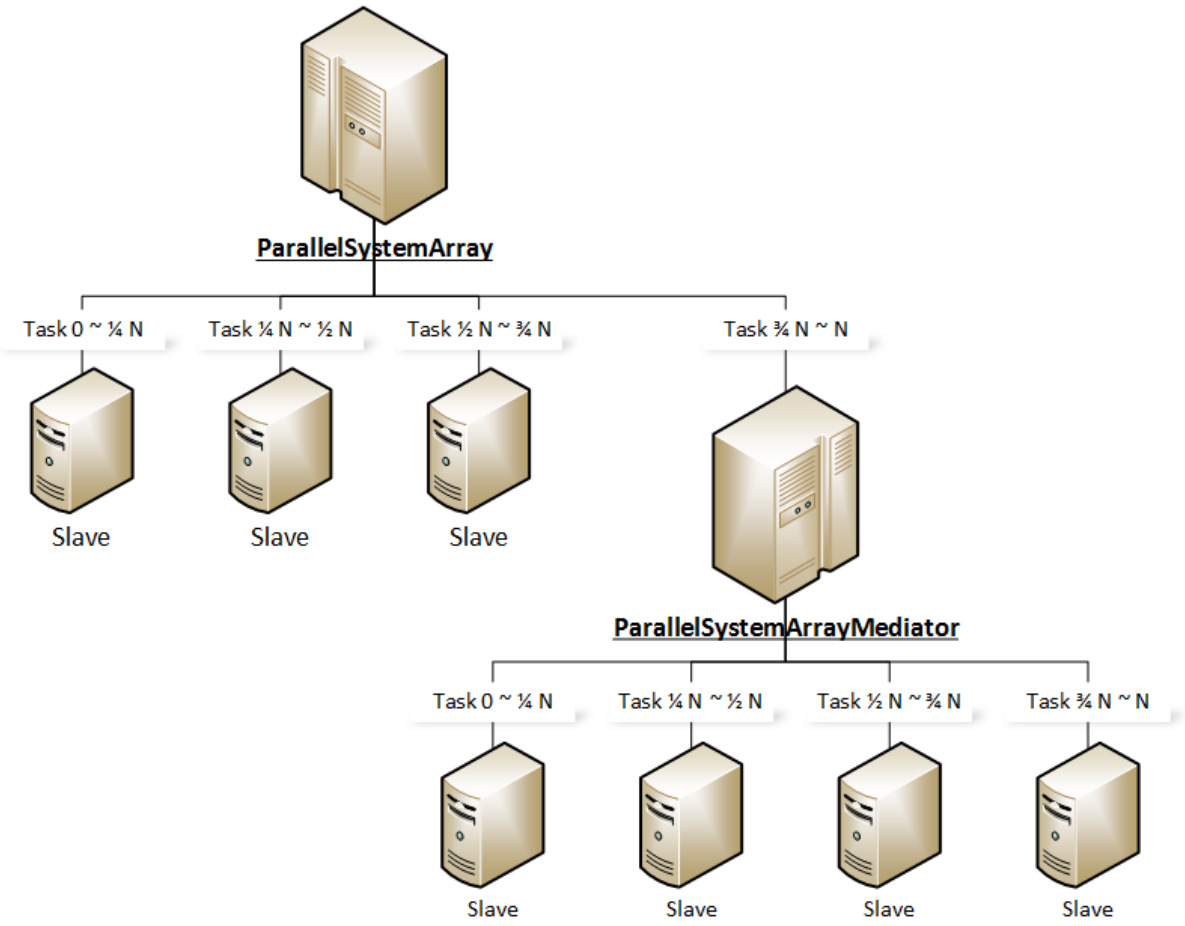
Additional Inherited Members

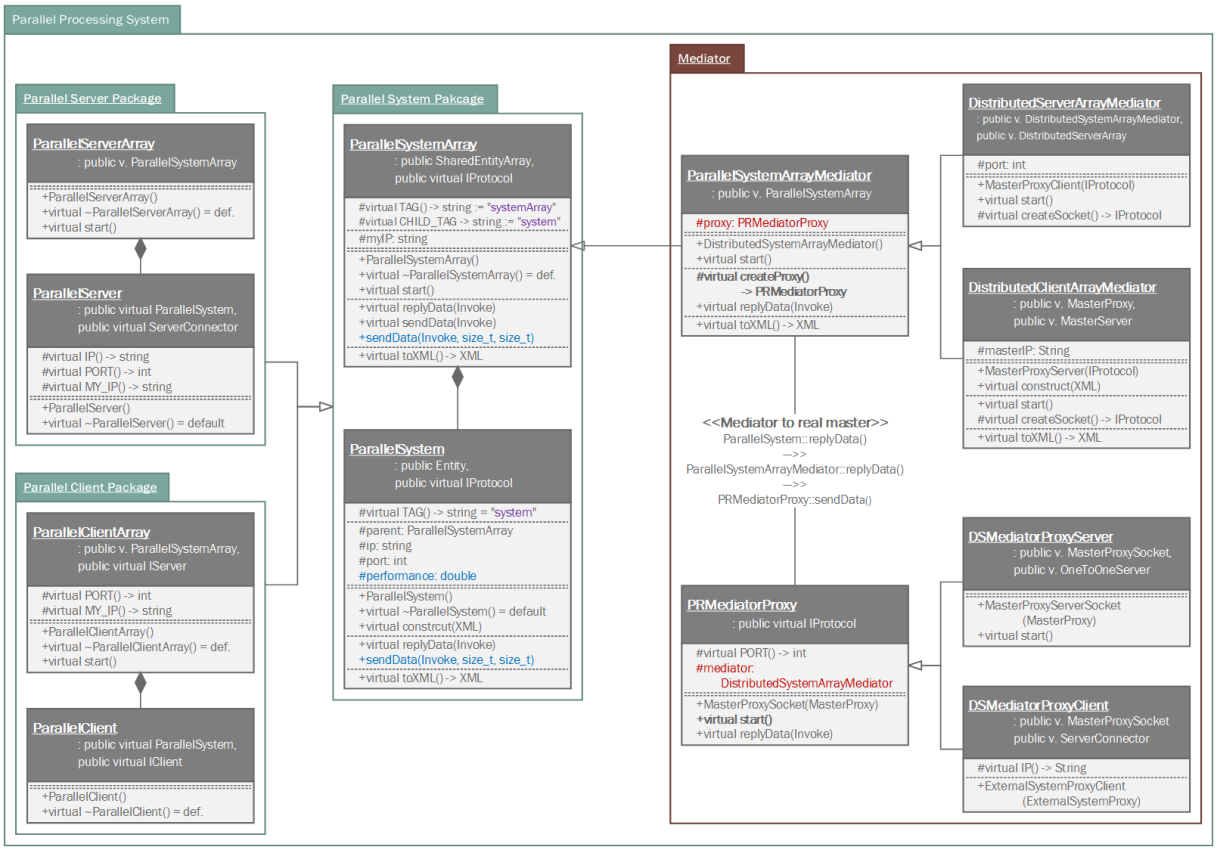
6.105.1 Detailed Description

A parallel slave system for mediation.

ParallelSlaveSystemMediator is an abstract class derived from slave::ParallelSystem for realizing tree-structured distriuted processins system. ParallelSlaveSystemMediator acts a slave system's role for its master, on the above level.

ParallelSystemArrayMediator and ParallelSlaveSystemMediator classes are paired to act a mediator system role by acting their own role, master system role for below slaves and slave system role for above master system.


ParallelSystemArray
Task 0 ~ ¼ N
Task ¼ N ~ ½ N
Task ½ N ~ ¾ N
Task ¾ N ~ N
Slave
Slave
Slave
ParallelSystemArrayMediator
Task 0 ~ ¼ N
Task ¼ N ~ ½ N
Task ½ N ~ ¾ N
Task ¾ N ~ N
Slave
Slave
Slave
Slave


Parallel Processing System
Parallel Server Package
ParallelServerArray
: public v. ParallelSystemArray
+ParallelServerArray()
+virtual ~ParallelServerArray() = def.
+virtual start()
ParallelServer
: public virtual ParallelSystem,
public virtual ServerConnector
#virtual IP() -> string
#virtual PORT() -> int
#virtual MY_IP() -> string
+ParallelServer()
+virtual ~ParallelServer() = default
Parallel Client Package
ParallelClientArray
: public v. ParallelSystemArray,
public virtual IServer
#virtual PORT() -> int
#virtual MY_IP() -> string
+ParallelClientArray()
+virtual ~ParallelClientArray() = def.
+virtual start()
ParallelClient
: public virtual ParallelSystem,
public virtual IClient
+ParallelClient()
+virtual ~ParallelClient() = def.
Parallel System Pakcage
ParallelSystemArray
: public SharedEntityArray,
public virtual IProtocol
#virtual TAG() -> string := "systemArray"
#virtual CHILD_TAG -> string := "system"
#myIP: string
+ParallelSystemArray()
+virtual ~ParallelSystemArray() = def.
+virtual start()
+virtual replyData(Invoke)
+virtual sendData(Invoke)
+sendData(Invoke, size_t, size_t)
+virtual toXML() -> XML
ParallelSystem
: public Entity,
public virtual IProtocol
#virtual TAG() -> string = "system"
#parent: ParallelSystemArray
#ip: string
#port: int
#performance: double
+ParallelSystem()
+virtual ~ParallelSystem() = default
+virtual constrcut(XML)
+virtual replyData(Invoke)
+sendData(Invoke, size_t, size_t)
+virtual toXML() -> XML
Mediator
ParallelSystemArrayMediator
: public v. ParallelSystemArray
#proxy: PRMediatorProxy
+DistributedSystemArrayMediator()
+virtual start()
#virtual createProxy()
-> PRMediatorProxy
+virtual replyData(Invoke)
+virtual toXML() -> XML
DistributedServerArrayMediator
: public v. DistributedSystemArrayMediator,
public v. DistributedServerArray
#port: int
+MasterProxyClient(IProtocol)
+virtual start()
#virtual createSocket() -> IProtocol
DistributedClientArrayMediator
: public v. MasterProxy,
public v. MasterServer
#masterIP: String
+MasterProxyServer(IProtocol)
+virtual construct(XML)
+virtual start()
#virtual createSocket() -> IProtocol
+virtual toXML() -> XML
<<Mediator to real master>>
ParallelSystem::replyData()
-->>
ParallelSystemArrayMediator::replyData()
-->>
PRMediatorProxy::sendData()
PRMediatorProxy
: public virtual IProtocol
#virtual PORT() -> int
#mediator:
DistributedSystemArrayMediator
+MasterProxySocket(MasterProxy)
+virtual start()
+virtual replyData(Invoke)
DSMediatorProxyServer
: public v. MasterProxySocket,
public v. OneToOneServer
+MasterProxyServerSocket
(MasterProxy)
+virtual start()
DSMediatorProxyClient
: public v. MasterProxySocket
public v. ServerConnector
#virtual IP() -> String
+ExternalSystemProxyClient
(ExternalSystemProxy)

[Inherited]

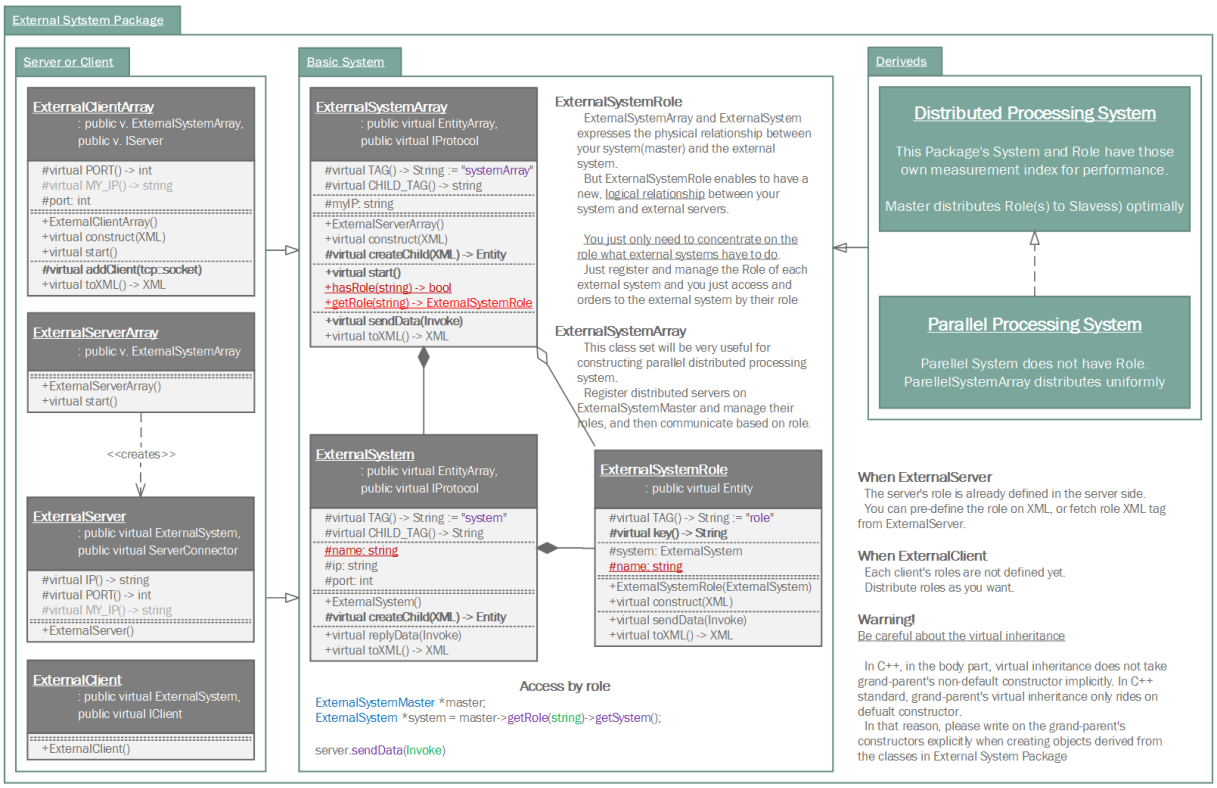
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::Parallel↩System and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystem↩Role.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an External↩SystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an External↩System. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

6.105.2 Member Function Documentation

6.105.2.1 virtual void replyData (std::shared_ptr< Invoke >) [override], [virtual]

Handling replied message from an external system.

If there's a related ExternalSystemRole, shifts responsibility to the ExternalSystemRole.

Parameters

invoke	Replied Invoke message

Reimplemented from ExternalSystem.

6.105.2.2 virtual auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Reimplemented from ExternalSystem.

The documentation for this class was generated from the following file:

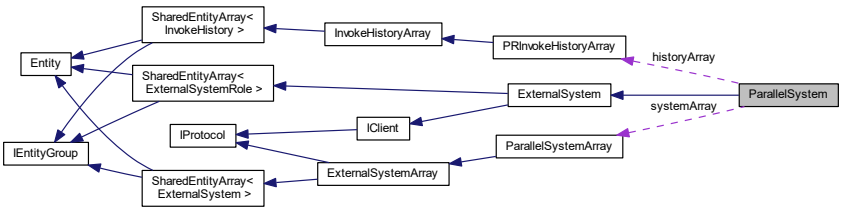
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelSlaveSystemMediator.↩ hpp

6.106 ParallelSystem Class Reference

A network driver for a parallel system.

`#include <ParallelSystem.hpp>`

Collaboration diagram for ParallelSystem:



Public Member Functions

- ParallelSystem ()
  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override
  Construct data of the Entity from an XML object.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the EntityGroup.

Protected Member Functions

- virtual void _replyData (std::shared_ptr< Invoke >) override

  A method for pre-processing replied Invoke message.

Protected Attributes

- PRInvokeHistoryArray * historyArray

  A list of history log for reported Invoke messages.

- double performance

  A performance index.

Private Member Functions

- void sendSegmentData (std::shared_ptr< Invoke >, size_t, size_t)

  Send a message with segmentation index.

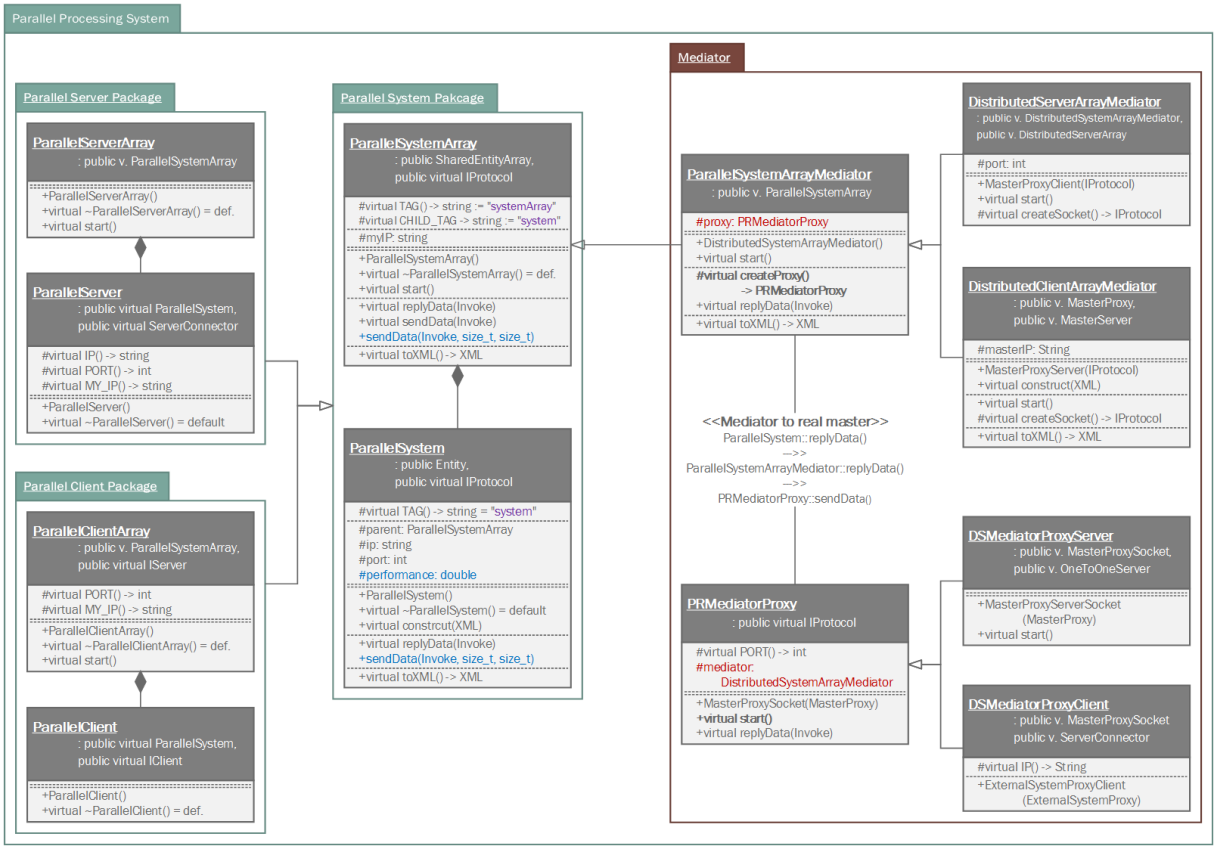
6.106.1 Detailed Description

A network driver for a parallel system.

ParallelSystem class is an ExternalSystem having performance index and history logs of handled Invoke messages with those elapsed times. Those performance index and history logs are used to determine how much segmentation to be allocated.

Each segmentation in requested process (Invoke message) is equivalent. Thus, role of ParallelSystem objects in a ParallelSystemArray are almost same and does not need to specify ExternalSystemRole on ParallelSystem.

- Not a matter to specifying ExternalSystemRole objects to each ParallelSystem. In that case, Invoke messages having segmentation size will be processed by ParallelSystem's own logic and Invoke messages without segmentation size will be handled by ordinary logic of ExternalSystem's own.



[Inherited]

6.106.2 Member Function Documentation

6.106.2.1 void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from ExternalSystem.

Reimplemented in ParallelServer.

6.106.2.2 virtual void _replyData (std::shared_ptr< Invoke >) [override], [protected], [virtual]

A method for pre-processing replied Invoke message.

You can realize a pre-processing about the replied Invoke message by overriding the IClient::_replyData().

Archiving a history log, filtering and truncating, let handling the message with a new thread, calculating performance by estimating elapsed time about the Invoke message, and so on. Anything you can define pre-processing works before IProtocol::replyData().

- Archive history log -> cloud service
- Estimating performance -> distributed and parallel processing system

Parameters

invoke	An invoke message to be pre-processed and shifted to IProtocol::replyData().

Reimplemented from IClient.

6.106.2.3 void sendSegmentData (std::shared_ptr< Invoke > , size_t , size_t) [private]

Send a message with segmentation index.

Parameters

invoke	An invoke message requesting a process.
startIndex	Starting index number of segmentation.
size	Size of segmentation.

6.106.2.4 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from ExternalSystem.

Reimplemented in ParallelServer.

6.106.3 Member Data Documentation

6.106.3.1 double performance [protected]

A performance index.

A performance index z is calculated by normalization calculating reverse number of

whole parallel system's average elapsed times and its z value between the normal distribution.

- X = (

  If a parallel system has no history of handling Invoke message, then set the performance index to 0 as default. The performance index will be re-calcuated whenever segmented process has sent (ParallelSystemArray::sendData(Invoke, size_t)).

  The documentation for this class was generated from the following files:

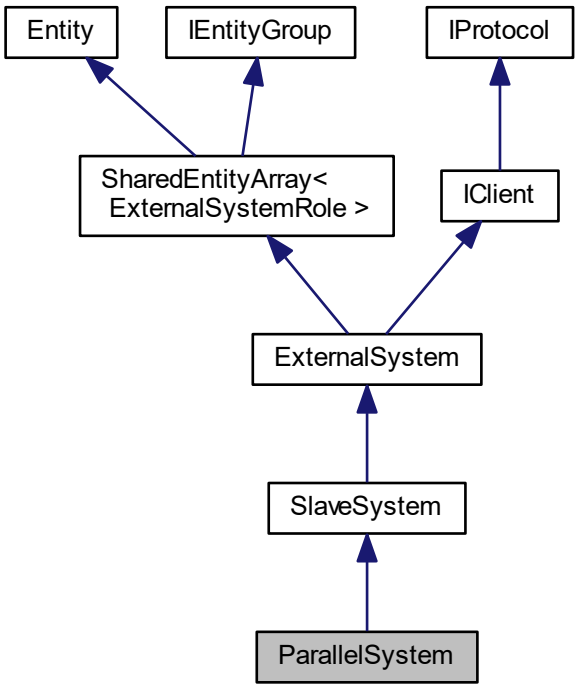
  * D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelSystem.hpp
  * D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelSystem.cpp

6.107 ParallelSystem Class Reference

A slave parallel system.

`#include <ParallelSystem.hpp>`

Collaboration diagram for ParallelSystem:



Public Member Functions

* ParallelSystem ()
  Default Constructor.

Protected Member Functions

* virtual void _replyData (std::shared_ptr< Invoke >) override
  Pre-processor for segmented processes and reporting elapsed time.
* virtual void replyData (std::shared_ptr< Invoke >, size_t, size_t)=0
  Handle requested processes which are segmented.

Additional Inherited Members

6.107.1 Detailed Description

A slave parallel system.

ParallelSystem is a class processing segmented Invoke messages from master.

[Inherited]

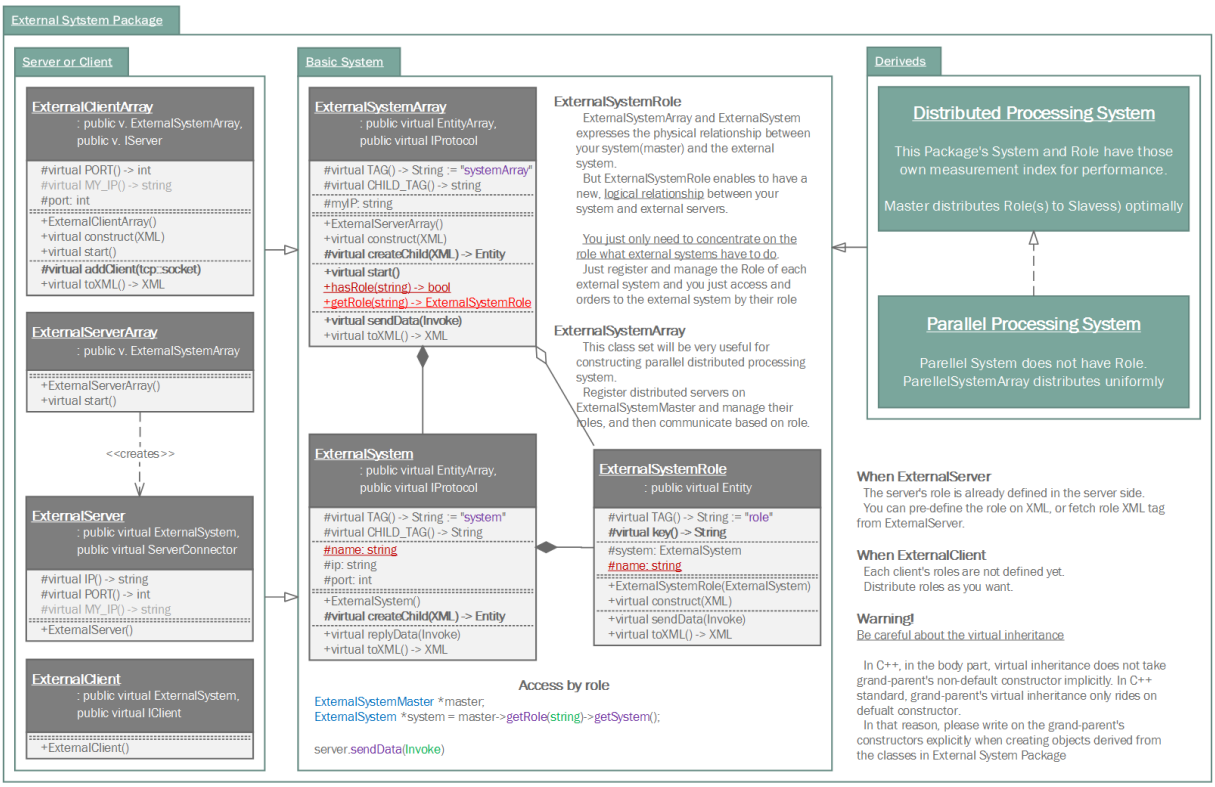
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::↩ ParallelSystem and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

6.107.2 Member Function Documentation

6.107.2.1 virtual void _replyData (std::shared_ptr< Invoke >) [override], [protected], [virtual]

Pre-processor for segmented processes and reporting elapsed time.

[Inherited]

You can realize a pre-processing about the replied Invoke message by overriding the IClient::_reply↩Data().

Archiving a history log, filtering and truncating, let handling the message with a new thread, calculating performance by estimating elapsed time about the Invoke message, and so on. Anything you can define pre-processing works before IProtocol::replyData().

* Archive history log -> cloud service
* Estimating performance -> distributed and parallel processing system

Parameters

invoke	An invoke message to be pre-processed and shifted to IProtocol::replyData().

Reimplemented from SlaveSystem.

6.107.2.2 virtual void replyData (std::shared_ptr< Invoke > , size_t , size_t) [protected], [pure virtual]

Handle requested processes which are segmented.

Parameters

invoke	An invoke message containing segmented process requestments.
startIndex	Starting index number of segmentation.
size	Size of segmentation.

The documentation for this class was generated from the following file:

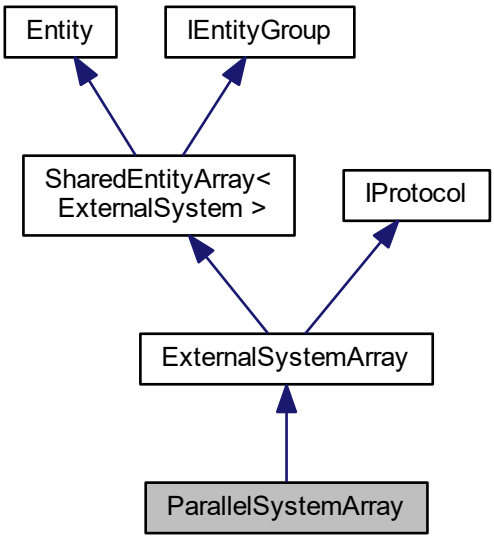
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/slave/ParallelSystem.hpp

6.108 ParallelSystemArray Class Reference

An array of parallel system drivers.

`#include <ParallelSystemArray.hpp>`

Collaboration diagram for ParallelSystemArray:



Public Member Functions

* ParallelSystemArray ()
  Default Constructor.
* virtual void sendSegmentData (std::shared_ptr< Invoke >, size_t)
  Send a message with segmentation size.

Additional Inherited Members

6.108.1 Detailed Description

An array of parallel system drivers.

ParallelSystemArray is an ExternalSystemArray and an abstract class containing. and managing parallel system drvers within framework of master. The ParallelSystemArray class allocates segmented processes (Invoke message containing segmentation size) following by each system's performance index.

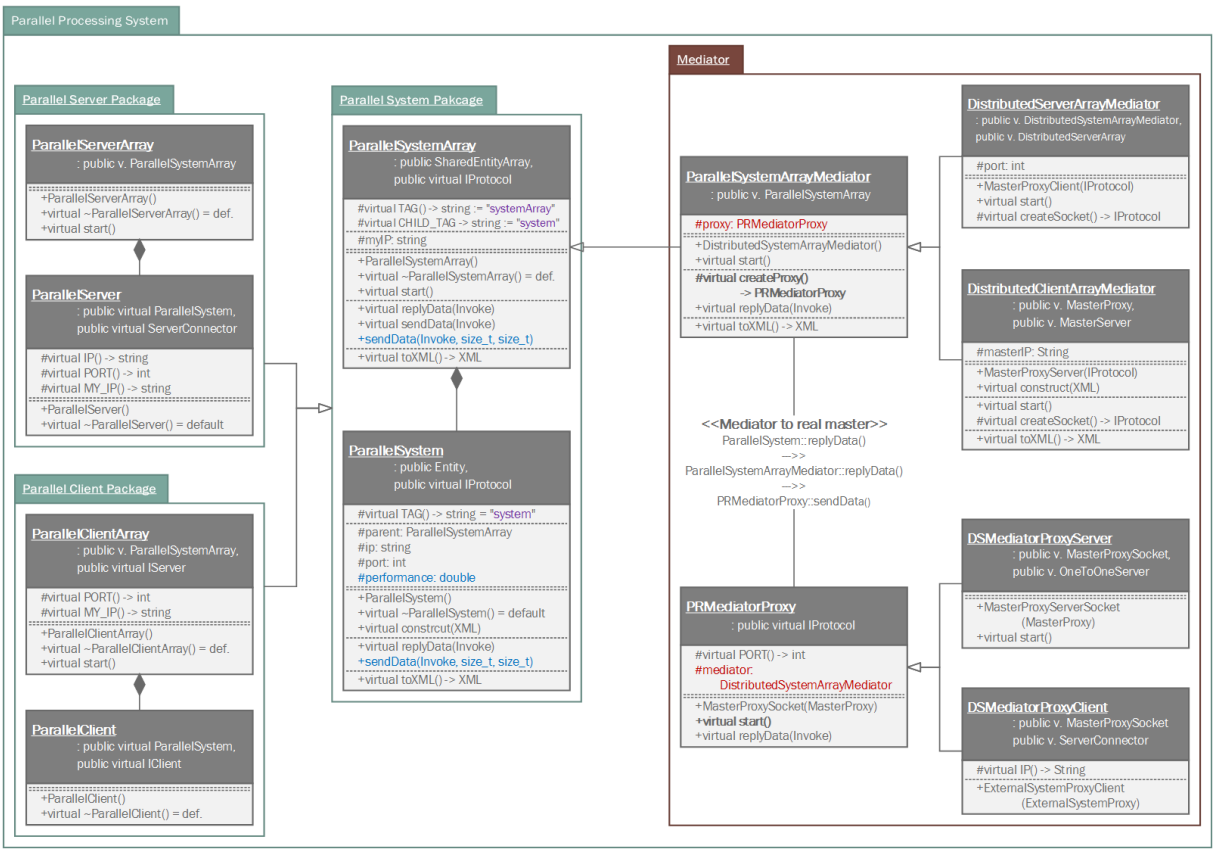
Unlike DistributedSystemArray, ParallelSystemArray does not have such complicate relationships and logics. All segmentation sent by sendData(Invoke, size_t) requires uniform performance. ParallelSystemArray just calculates each system's performance index by elapsing time such easily.

Of course, DistributedSystemArray determines which size of segmentation allocation will be suitable for each system by the performance index

.

Each segmentation in requested process (Invoke message) is equivalent. Thus, role of ParallelSystem objects in a ParallelSystemArray are almost same and does not need to specify ExternalSystemRole on each ParallelSystem.

* Not a matter to specifying ExternalSystemRole objects to each ParallelSystem. In that case, Invoke messages having segmentation size will be processed by ParallelSystemArray's own logic and Invoke messages without segmentation size will be handled by ordinary logic of ExternalSystemArray's own.



[Inherited]

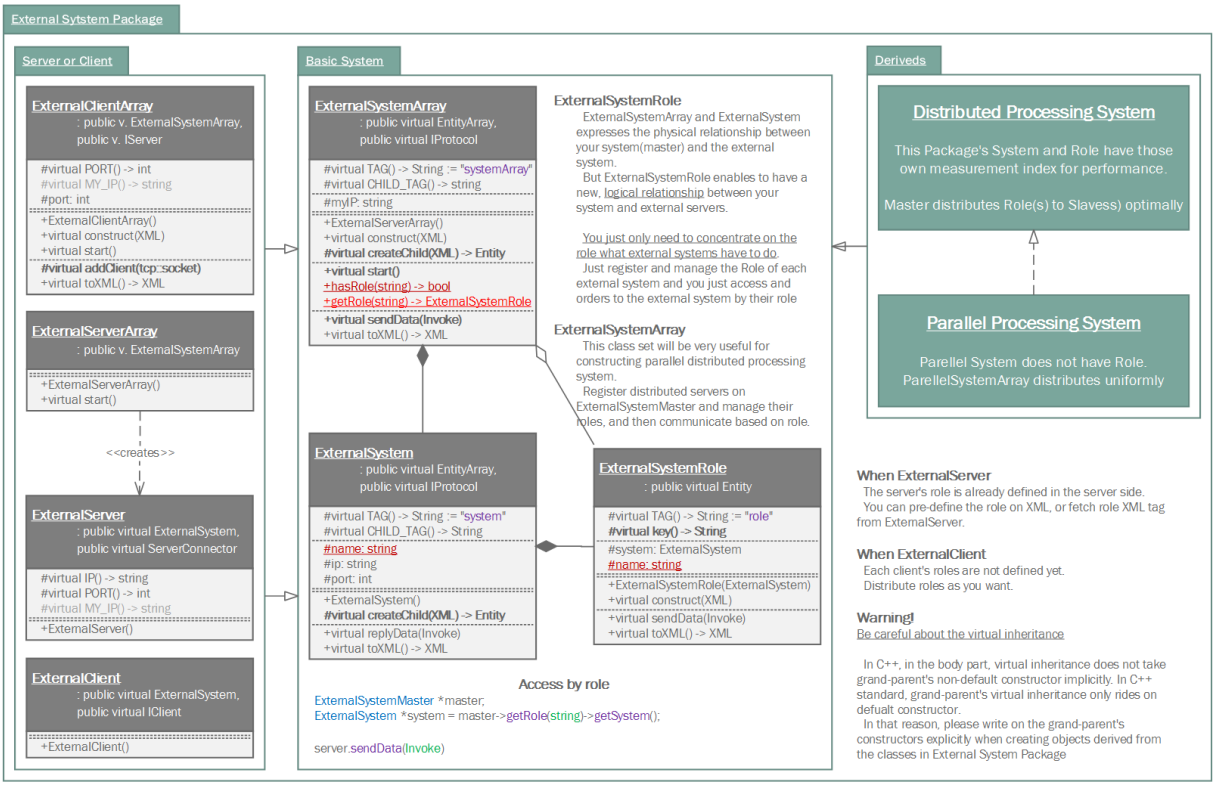
ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole

is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an External↩ SystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which External↩ System.

* ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.108.2 Member Function Documentation

6.108.2.1 void sendSegmentData (std::shared_ptr< Invoke >, size_t) [virtual]

Send a message with segmentation size.

Parameters

invoke	An invoke message requesting a process.
size	Size of segmentation.

The documentation for this class was generated from the following files:

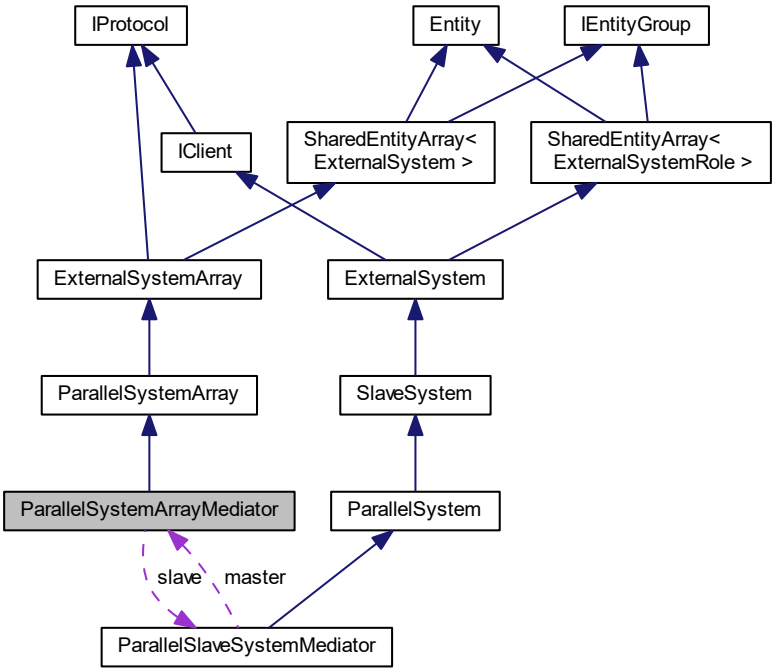
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelSystemArray.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelSystemArray.cpp

6.109 ParallelSystemArrayMediator Class Reference

A mediator of parallel system between master and slaves.

```
#include <ParallelSystemArrayMediator.hpp>
```

Collaboration diagram for ParallelSystemArrayMediator:



Public Member Functions

- ParallelSystemArrayMediator ()
  Default Constructor.
- virtual void construct (std::shared_ptr< library::XML >) override
  Construct data of the Entity from an XML object.
- virtual void start () override
  Start interaction.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the EntityGroup.

Protected Member Functions

- virtual auto createSlave () const -> ParallelSlaveSystemMediator *=0
  Factory method of slave system for mediation.

Protected Attributes

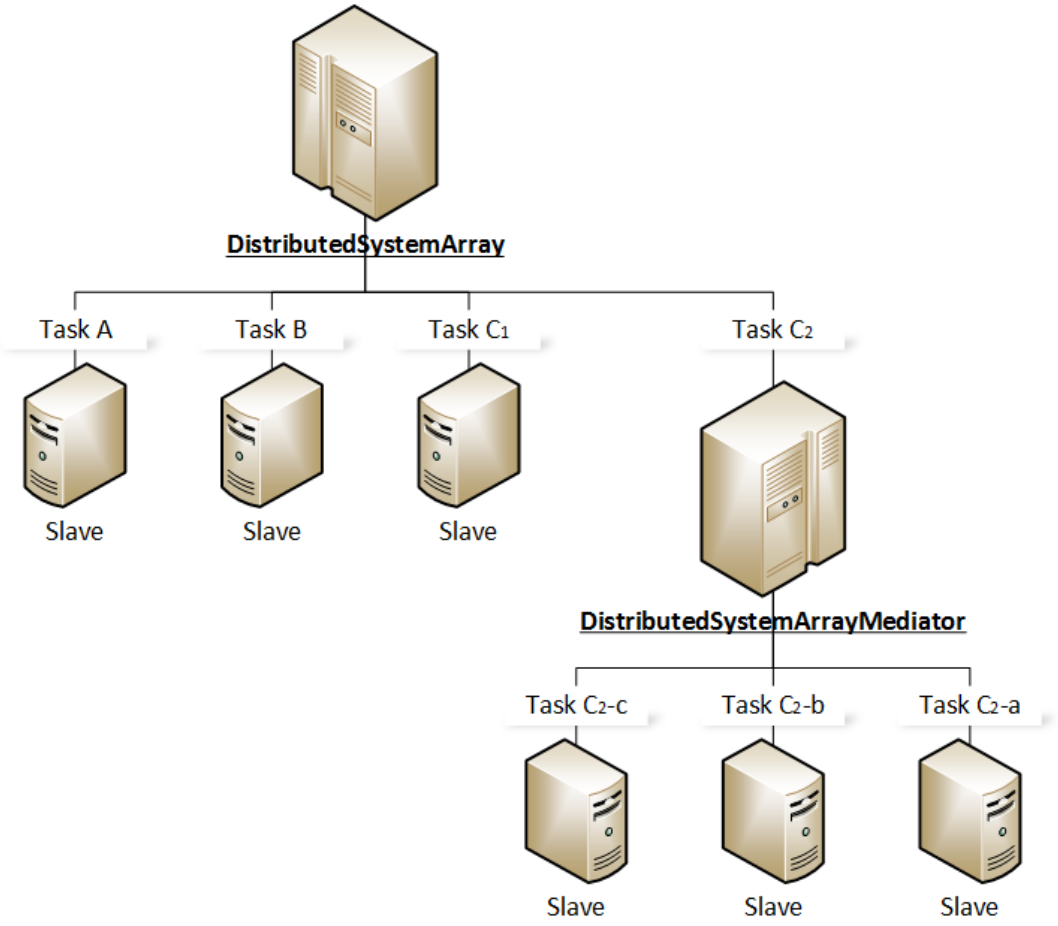
- ParallelSlaveSystemMediator * slave
  A slave system for mediation.

6.109.1 Detailed Description

A mediator of parallel system between master and slaves.

ParallelSystemArrayMediator class is used to realizing a mediator system of tree-structured parallel processing system. ParallelSystemArrayMediator intermediates between origin-master(ParallelSystem↩Array) and its slaves(slave::ParallelSystem).

ParallelSystemArrayMediator is a master(ParallelSystemArray) within the framework of slaves systems (slave::ParallelSystem objects) and also can be a slave system (slave::ParallelSystem) within framework of its master(ParallelSystemArray) by having ParallelSlaveSystemMediator.



[Inherited]

ParallelSystemArray is an ExternalSystemArray and an abstract class containing. and managing parallel system drvers within framework of master. The ParallelSystemArray class allocates segmented processes (Invoke message containing segmentation size) following by each system's performance index.

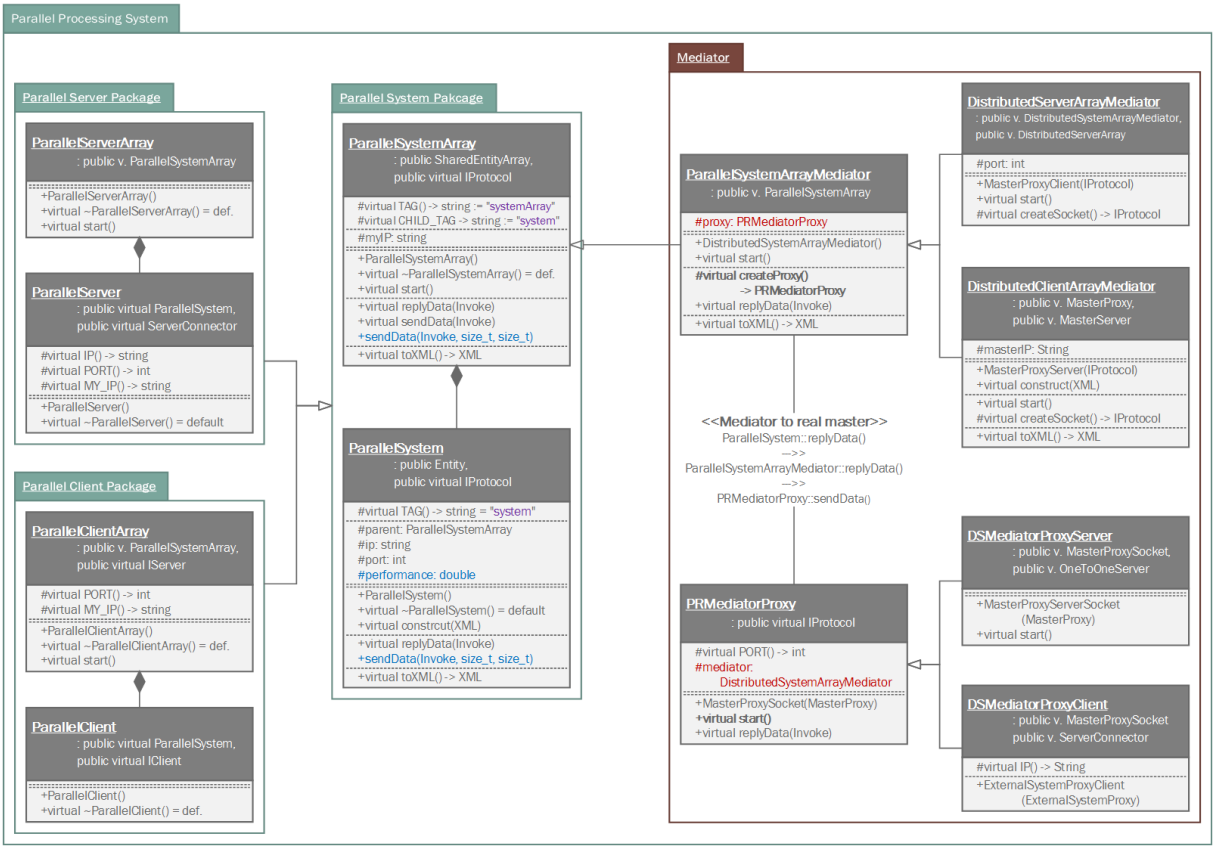
Unlike DistributedSystemArray, ParallelSystemArray does not have such complicate relationships and logics. All segmentation sent by sendData(Invoke, size_t) requires uniform performance. ParallelSystemArray just calculates each system's performance index by elapsing time such easily.

Of course, DistributedSystemArray determines which size of segmentation allocation will be suitable for each system by the performance index

.

Each segmentation in requested process (Invoke message) is equivalent. Thus, role of ParallelSystem objects in a ParallelSystemArray are almost same and does not need to specify ExternalSystemRole on each ParallelSystem.

* Not a matter to specifying ExternalSystemRole objects to each ParallelSystem. In that case, Invoke messages having segmentation size will be processed by ParallelSystemArray's own logic and Invoke messages without segmentation size will be handled by ordinary logic of ExternalSystemArray's own.



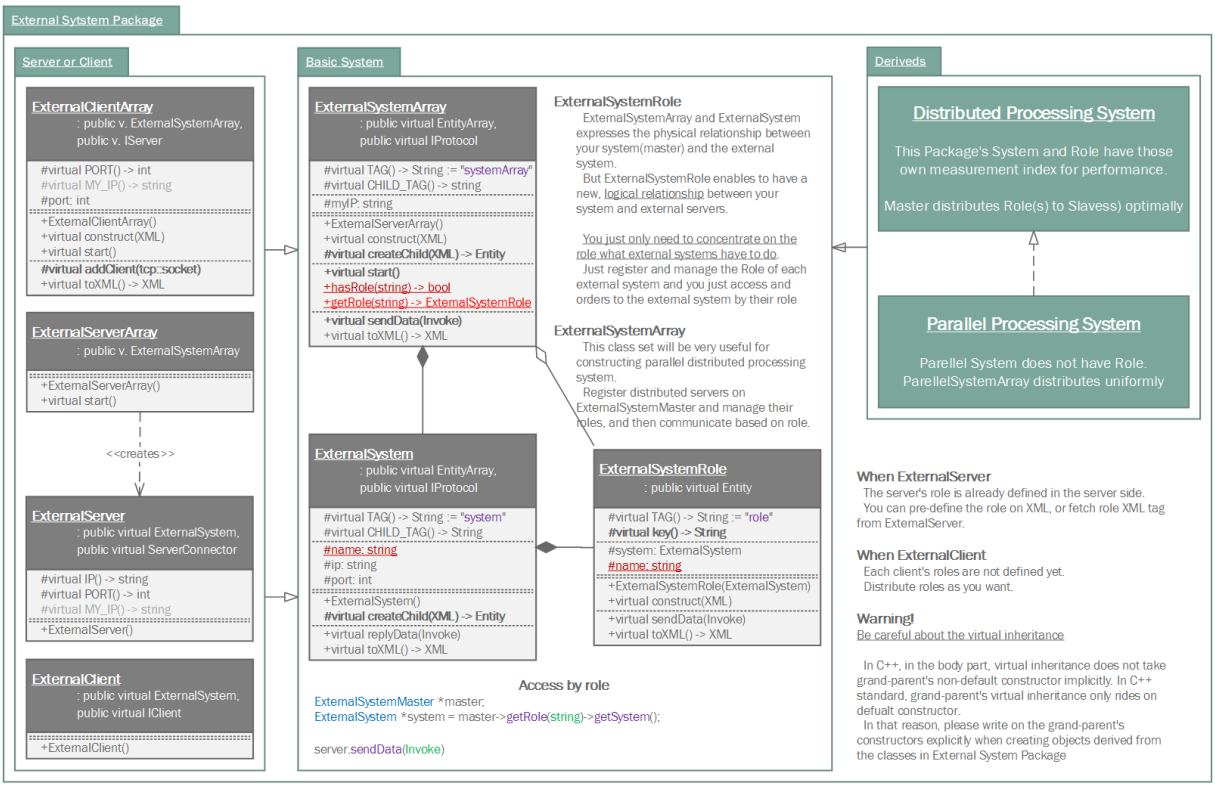
[Inherited]

ExternalSystemArray is an abstract class containing and managing external system drivers.

Also, ExternalSystemArray can access to ExternalSystemRole(s) directly. With the method, you can use an ExternalSystemRole as "logical proxy" of an ExternalSystem. Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an External↩ SystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, the ExternalSystemRole acted a role of proxy.

It's called as "Proxy pattern". With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which External↩ System.

* ExternalSystemArray::getRole("something")->sendData(invoke);



See also

samchon::protocol

Author

Jeongho Nam

6.109.2 Member Function Documentation

6.109.2.1 virtual void construct (std::shared_ptr< library::XML > *xml*) [override], [virtual]

Construct data of the Entity from an XML object.

Constructs the EntityGroup's own member variables only from the input XML object.

Do not consider about constructing children Entity objects' data in EntityGroup::construct(). Those children Entity objects' data will constructed by their own construct() method. Even insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Constructs only data of EntityGroup's own.

[Inherited]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in ParallelClientArrayMediator.

6.109.2.2 virtual auto createSlave ( ) const -> ParallelSlaveSystemMediator * [protected], [pure virtual]

Factory method of slave system for mediation.

Exceptions

invalid_parameter	If ParallelServerArrayMediator, slave must be a type of ParallelSlaveServer↩ Mediator

6.109.2.3 virtual void start ( ) [override], [virtual]

Start interaction.

An abstract method starting interaction with external systems.

If external systems are servers, starts connection to them, else clients, opens a server and accepts the external systems. You can addict your own procudures of starting drivers, but if you directly override method of abstract ExternalSystemArray, be careful about virtual inheritance.

* DistributedClientArray::start() calls DistributedSystemArray::start() and ExternalClientArray:start()

Implements ExternalSystemArray.

Reimplemented in ParallelClientArrayMediator, and ParallelServerArrayMediator.

6.109.2.4 virtual auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the EntityGroup.

Archives the EntityGroup's own member variables only to the returned XML object.

Do not consider about archiving children Entity objects' data in EntityGroup::toXML(). Those children Entity objects will converted to XML object by their own toXML() method. The insertion of XML objects representing children are done by abstract method of EntityGroup::toXML().

Archives only data of EntityGroup's own.

[Inherited]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from EntityGroup< _Container, _ETy, _Ty >.

Reimplemented in ParallelClientArrayMediator.

The documentation for this class was generated from the following file:

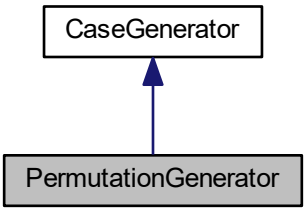
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/ParallelSystemArray↩ Mediator.hpp

6.110 PermutationGenerator Class Reference

A permutation case generator.

```
#include <PermutationGenerator.hpp>
```

Collaboration diagram for PermutationGenerator:



Public Member Functions

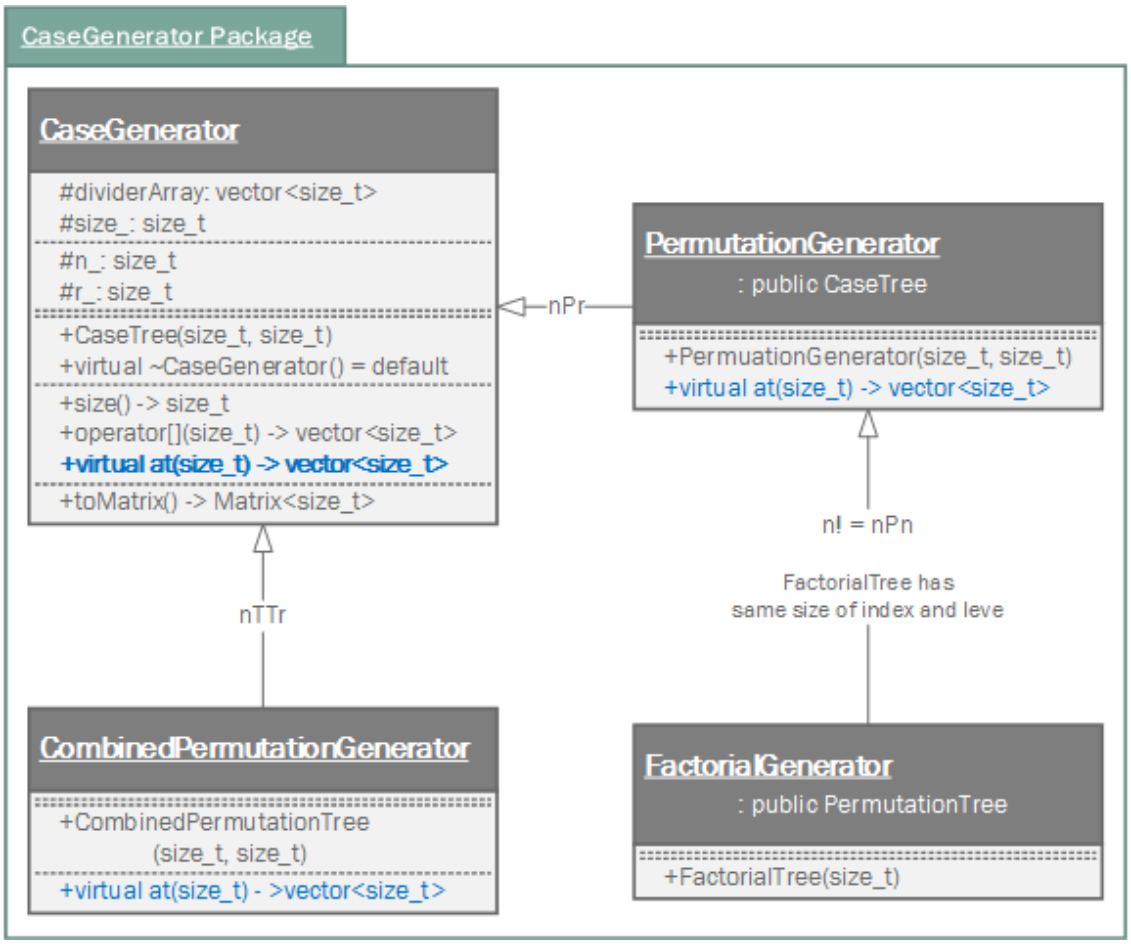
* PermutationGenerator (size_t n, size_t r)
  Construct from size of N and R.
* virtual auto operator[ ] (size_t) const -> std::vector< size_t > override
  Get x'th case.

Additional Inherited Members

6.110.1 Detailed Description

A permutation case generator.

nPr



See also

samchon::library

Author

Jeongho Nam

6.110.2 Constructor & Destructor Documentation

6.110.2.1 PermutationGenerator (size_t *n*, size_t *r*)

Construct from size of N and R.

Parameters

n	Size of candidates
r	Size of elements of each case

6.110.3 Member Function Documentation

6.110.3.1 auto operator[] (size_t) const -> std::vector< size_t > [override], [virtual]

Get x'th case.

Returns

The row of the x'th in combined permuation case

Implements CaseGenerator.

The documentation for this class was generated from the following files:

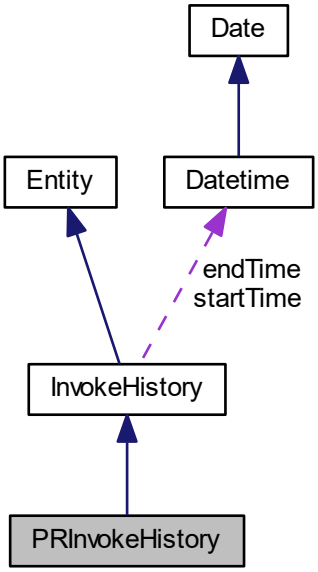
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/PermutationGenerator.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/PermutationGenerator.cpp

6.111 PRInvokeHistory Class Reference

A reported history of an Invoke message.

`#include <PRInvokeHistory.hpp>`

Collaboration diagram for PRInvokeHistory:



Public Member Functions

- PRInvokeHistory ()
  Default Constructor.
- PRInvokeHistory (std::shared_ptr< Invoke >)
  Construct from an Invoke message.
- virtual void construct (std::shared_ptr< library::XML >) override
  Construct data of the Entity from an XML object.
- auto getIndex () const -> size_t
  Get index.
- auto getSize () const -> size_t
  Get size.
- virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the Entity.

Protected Attributes

- size_t index
  Start index.
- size_t size
  The Size.

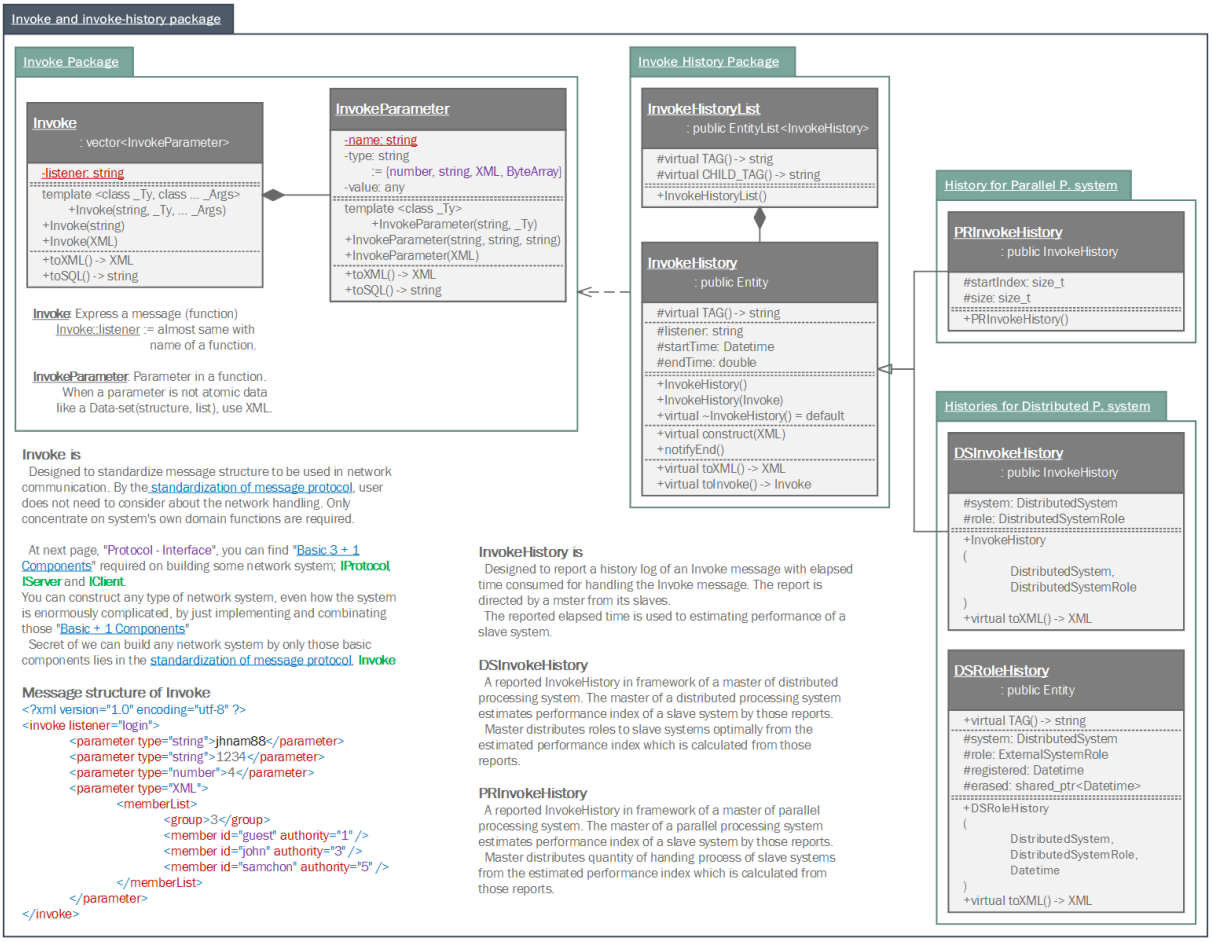
6.111.1 Detailed Description

A reported history of an Invoke message.

[Inherited]

InvokeHistory is a class for reporting history log of an Invoke message with elapsed time from a slave to its master.

With the elapsed time, consumed time for a process of handling the Invoke message, InvokeHistory is reported to the master. The master utilizies the elapsed time to estimating performances of each slave system. With the estimated performan index, master retrives the optimal solution of distributing processes.



See also

samchon::protocol
samchon::protocol::master
samchon::protocol::slave

Author

Jeongho Nam

6.111.2 Constructor & Destructor Documentation

6.111.2.1 PRInvokeHistory (std::shared_ptr< Invoke >)

Construct from an Invoke message.

InvokeHistory does not archive entire Invoke message, only archives its listener.

Parameters

invoke	A message to archive its history log

6.111.3 Member Function Documentation

6.111.3.1 void construct (std::shared_ptr< library::XML >) [override],[virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Reimplemented from InvokeHistory.

6.111.3.2 auto toXML ( ) const -> std::shared_ptr<library::XML> [override],[virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from InvokeHistory.

The documentation for this class was generated from the following files:

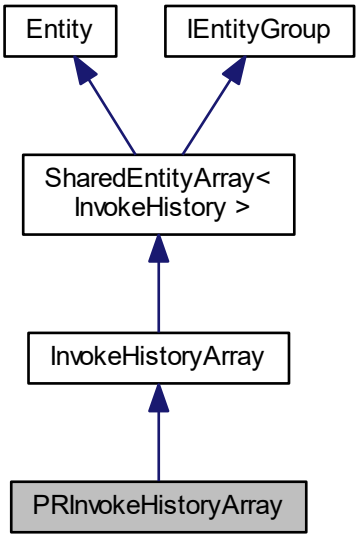
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/PRInvokeHistory.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/PRInvokeHistory.cpp

6.112 PRInvokeHistoryArray Class Reference

An array of PRInvokeHistory.

```
#include <PRInvokeHistoryArray.hpp>
```

Collaboration diagram for PRInvokeHistoryArray:



Public Member Functions

* PRInvokeHistoryArray ()
  Default Constructor.

Protected Member Functions

* virtual auto createChild (std::shared_ptr< library::XML >) -> InvokeHistory *override
  Factory method of a child Entity.

6.112.1 Detailed Description

An array of PRInvokeHistory.

Author

Jeongho Nam

6.112.2 Member Function Documentation

6.112.2.1 auto createChild (std::shared_ptr< library::XML >) -> InvokeHistory *override

`[override], [protected], [virtual]`

Factory method of a child Entity.

EntityGroup::createChild() is a factory method creating a new child Entity which is belonged to the EntityGroup. This method is called by EntityGroup::construct(). The children construction methods Entity::construct() will be called by abstract method of the EntityGroup::construct().

Returns

A new child Entity belongs to EntityGroup.

Reimplemented from InvokeHistoryArray.

The documentation for this class was generated from the following files:

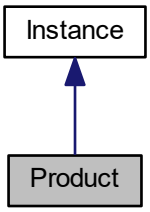
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/PRInvokeHistoryArray.↩hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/master/PRInvokeHistoryList.cpp

6.113 Product Class Reference

A product, merchandise.

`#include <Product.hpp>`

Collaboration diagram for Product:



Public Member Functions

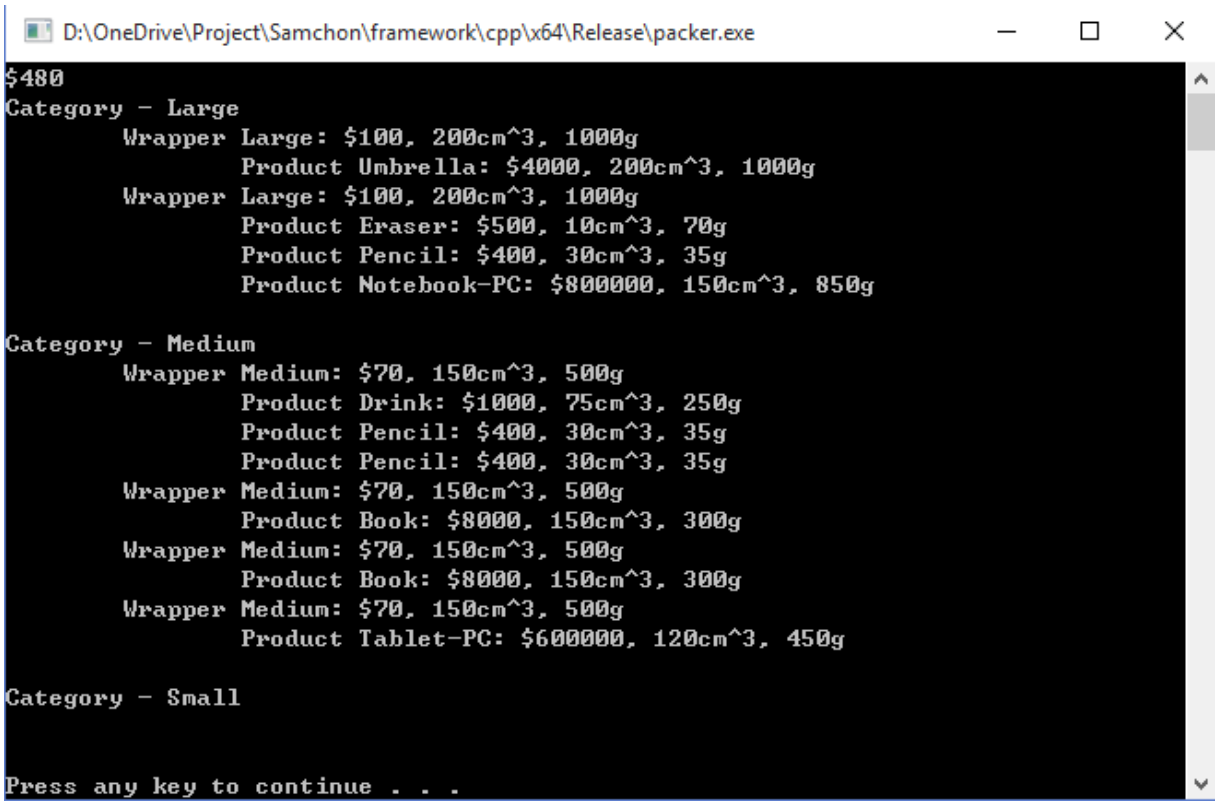
* Product (const std::string &name, int price, int volume, int weight)
  Construct from arguments.
* virtual auto toString () const -> std::string override
  Return a string represents the Instance.

Additional Inherited Members

6.113.1 Detailed Description

A product, merchandise.

A merchandise with its price, volume and weight.



Author

Jeongho Nam

6.113.2 Constructor & Destructor Documentation

6.113.2.1 Product (const std::string & *name*, int *price*, int *volume*, int *weight*)

Construct from arguments.

Parameters

name	Name of a product can be an identifier
price	Price of the product
volume	Volume of the product
weight	Weight of the product

The documentation for this class was generated from the following files:

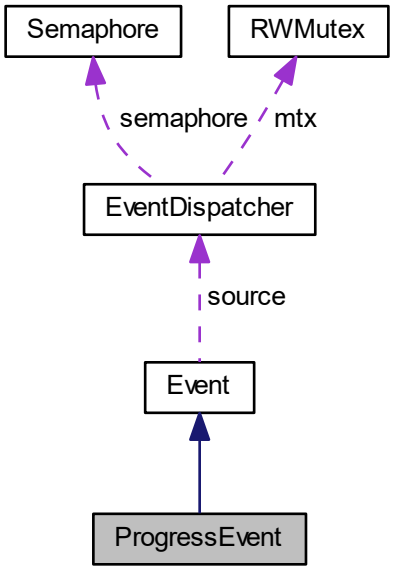
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/Product.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/Product.cpp

6.114 ProgressEvent Class Reference

Event representing a progress.

```
#include <ProgressEvent.hpp>
```

Collaboration diagram for ProgressEvent:



Public Member Functions

- ProgressEvent (EventDispatcher *, size_t, size_t)
  Construct from source and progress.
- auto getNumerator () const -> size_t
  Get numerator, number of current progress.
- auto getDenominator () const -> size_t
  Get denominator, number of total progress.
- auto getPercent () const -> double
  Get percentage of the progress.

Protected Attributes

- size_t numerator
  The number of current progress.
- size_t denominator
  The number of total progress.

6.114.1 Detailed Description

Event representing a progress.

[Inherited]

6.114.2 Constructor & Destructor Documentation

6.114.2.1 ProgressEvent (EventDispatcher * *source*, size_t *numerator*, size_t *denominator*)

Construct from source and progress.

Parameters

source	Source of the event; who made the event
numerator	The number of current progress
denominator	The number of total progress

Exceptions

exception	denominator is zero

6.114.3 Member Data Documentation

6.114.3.1 size_t denominator [protected]

The number of total progress.

Warning

denominator cannot be zero

The documentation for this class was generated from the following files:

* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/ProgressEvent.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/ProgressEvent.cpp

6.115 RWMutex Class Reference

rw_mutex

```
#include <RWMutex.hpp>
```

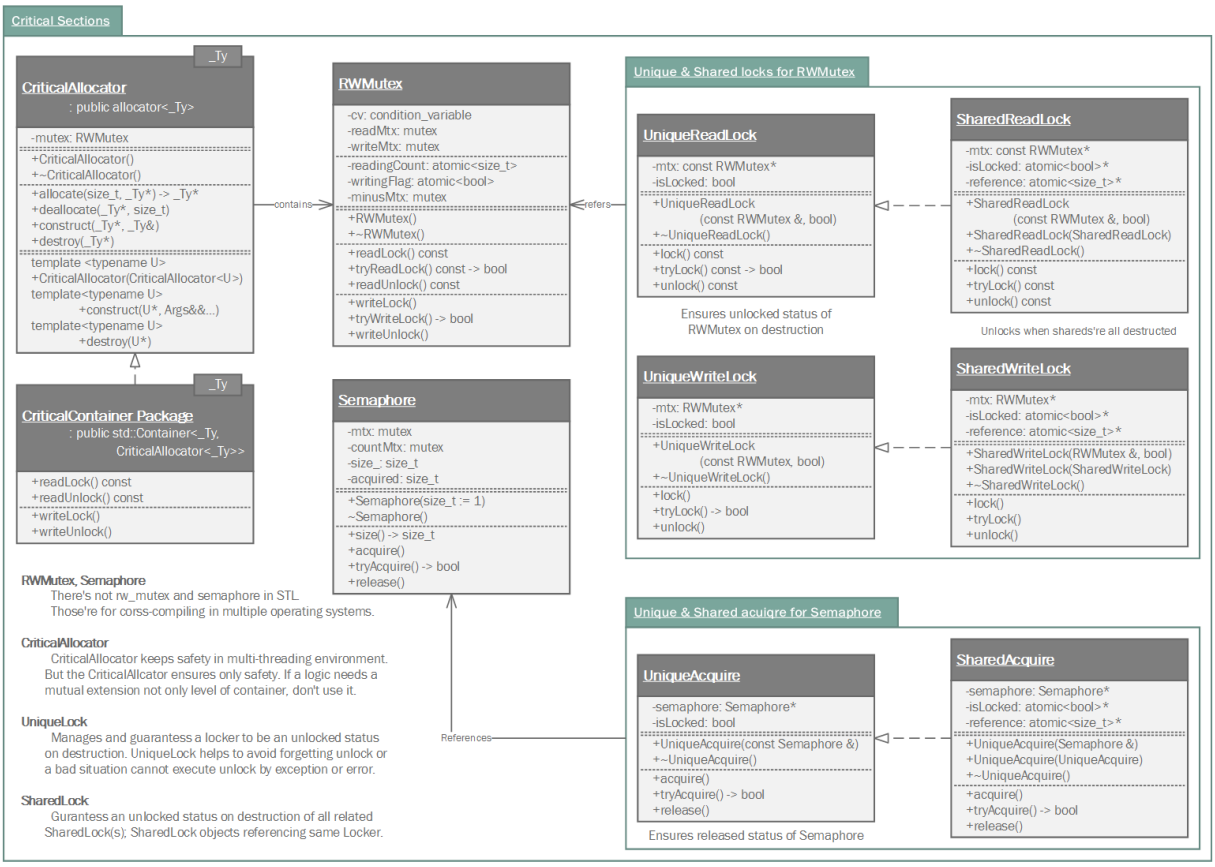
Public Member Functions

* RWMutex ()
  Default Constructor.
* void readLock () const
  Lock on read.
* void readUnlock () const
  Unlock of read.
* void writeLock ()
  Lock on writing.
* void writeUnlock ()
  Unlock on writing.

6.115.1 Detailed Description

rw_mutex

A mutex divided into reading and writing.



Note

Of course, rw_mutex is already defined in linux C. But it is dependent on the linux OS, so that cannot be compiled in Window having the rw_mutex. There's not a class like rw_mutex in STL yet. It's the reason why RWMutex is provided.

As that reason, if STL supports the rw_mutex in near future, the RWMutex can be depreciated.

See also

samchon::library

Author

Jeongho Nam

6.115.2 Member Function Documentation

6.115.2.1 void readLock ( ) const

Lock on read.

Increases a reading count.

When write_lock is on a progress, wait until write_unlock to be called.

* Reading can be done by multiple sections.
* Reading can't be done when writing.

Warning

You've to call read_unlock when the reading work is terminated.

6.115.2.2 void readUnlock ( ) const

Unlock of read.

Decreases a reading count.

When write_lock had done after read_lock, it continues by read_unlock if the reading count was 1 (read_unlock makes the count to be zero).

6.115.2.3 void writeLock ( )

Lock on writing.

Changes writing flag to true.

If another write_lock or read_lock is on a progress, wait until them to be unlocked.

* Writing can be done by only a section at once.
* Writing can't be done when reading.

Note

You've to call write_unlock when writing work was terminated.

The documentation for this class was generated from the following files:

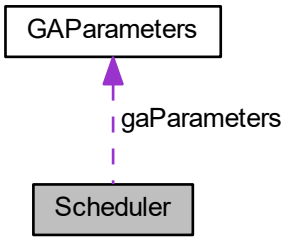
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/RWMutex.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/RWMutex.cpp

6.116 Scheduler Class Reference

A scheduler.

`#include <Scheduler.hpp>`

Collaboration diagram for Scheduler:



Public Member Functions

* Scheduler (std::shared_ptr< Travel >, const struct GAParameters &)
  Construct from points and parameter of genetic algorithm.
* auto optimize () -> std::shared_ptr< Travel >
  Derive optimized schedule.

Private Attributes

* std::shared_ptr< Travel > travel
  A travel to optimize or optimized.
* struct GAParameters gaParameters
  Parameters for Genetic-Algorithm.

6.116.1 Detailed Description

A scheduler.

Scheduler deducts an optimal schdule(Travel) by genetic algorithm.

Author

Jeongho Nam

6.116.2 Member Function Documentation

6.116.2.1 auto optimize ( ) -> std::shared_ptr<Travel>

Derive optimized schedule.

Returns

An optimized travel schedule.

The documentation for this class was generated from the following files:

* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/tsp/Scheduler.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/tsp/Scheduler.cpp

6.117 Semaphore Class Reference

A semaphore.

```
#include <Semaphore.hpp>
```

Public Member Functions

* Semaphore (size_t=2)
* void setSize (size_t)
  Set size.
* auto size () const -> size_t
  Get size.
* auto acquiredSize () const -> size_t
  Get acquired size.
* void acquire ()
  Acquire admission.
* auto tryAcquire () -> bool
  Try to acquire admission.
* void release ()
  Release an admission.

Private Attributes

* size_t size_
  The size.
* size_t acquired
  Acquired count.
* std::mutex * mtx
  Locker.

6.117.1 Detailed Description

A semaphore.

In computer science, particularly in operating systems, a semaphore is a variable or abstract data type that is used for controlling access, by multiple processes, to a common resource in a concurrent system such as a multiprogramming operating system.

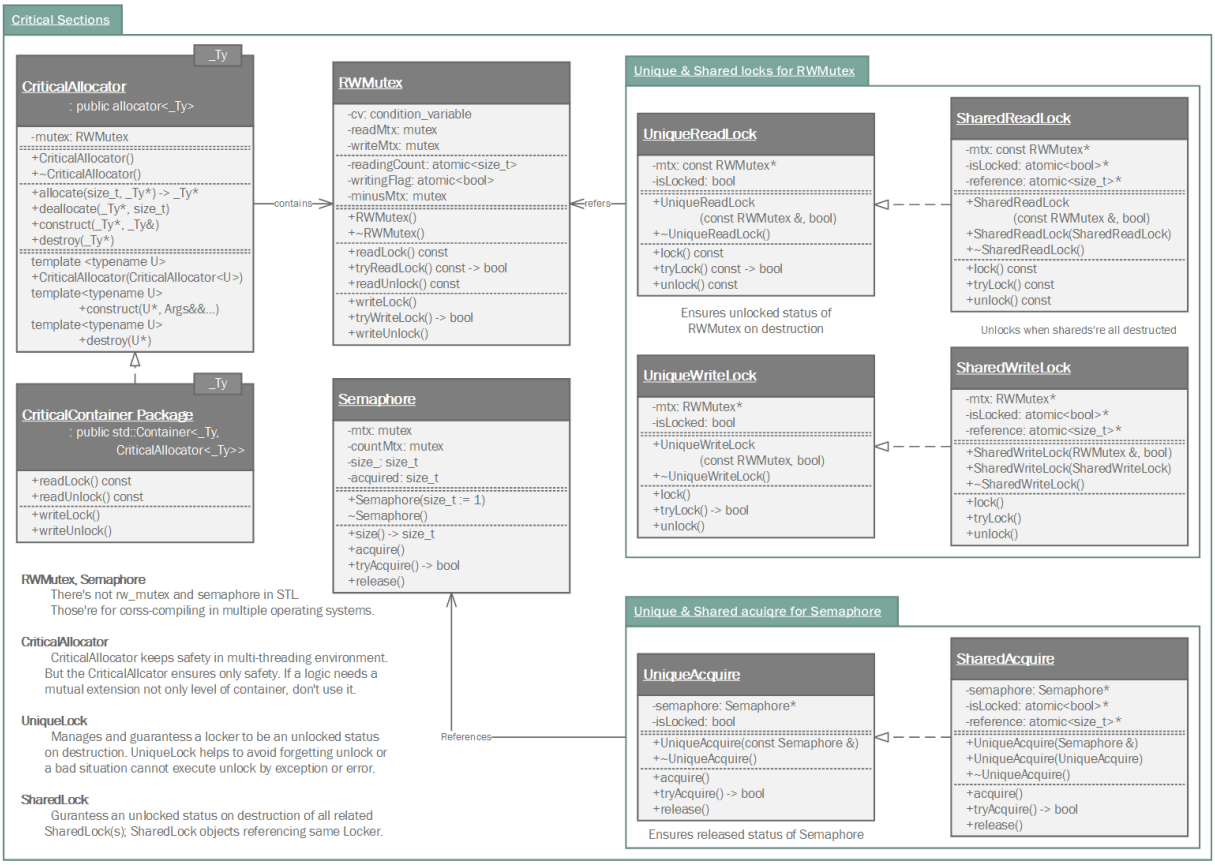
A trivial semaphore is a plain variable that is changed (for example, incremented or decremented, or toggled) depending on programmer-defined conditions. The variable is then used as a condition to control access to some system resource.

A useful way to think of a semaphore as used in the real-world systems is as a record of how many units of a particular resource are available, coupled with operations to safely (i.e., without race conditions) adjust that record as units are required or become free, and, if necessary, wait until a unit of the resource becomes available. Semaphores are a useful tool in the prevention of race conditions; however, their use is by no means a guarantee that a program is free from these problems. Semaphores which allow an arbitrary resource count are called counting semaphores, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores

* Categorized in binary semaphore

Referenced Wediapedia

* https://en.wikipedia.org/wiki/Semaphore_(programming)



Note

Of course, semaphore is already defined in linux C and MFC in Window. But it is dependent on each operating system, so that cannot be compiled in another OS with those semaphores. There's not a class like semaphore in STL yet. It's the reason why Semaphore is provided.

As that reason, if STL supports the semaphore in near future, the Semaphore can be depreciated.

See also

samchon::library

Author

Jeongho Nam

6.117.2 Constructor & Destructor Documentation

6.117.2.1 Semaphore (size_t *size* = 2)

Constructor.

Parameters

size	The size of the semaphore to permit

6.117.3 Member Function Documentation

6.117.3.1 void setSize (size_t *val*)

Set size.
Set permitted size of the semaphore.

6.117.3.2 auto size ( ) const -> size_t

Get size.
Returns size which means the permitted count of the semaphore
Returns

The size of semaphore

6.117.3.3 void acquire ( )

Acquire admission.
Acquires an admission and increases count of admission by 1.
If the count is over permitted size, wait until other admissions to be released.

* Lock on mutex

6.117.3.4 auto tryAcquire ( ) -> bool

Try to acquire admission.
If admission count is below the permitted size, acquire admission and increase the count by 1 and return true which means succeded to get admission.

* If the count is matched to the permitted size, lock the mutex

Else, do not acquire admission and return false which means failed to get admmission.
Returns

Whether succeded to acquire an admission or not

6.117.3.5 void release ( )

Release an admission.
Releases an admission what you've acquired. If the admission count was over the limited size, unlock the mutex.

6.117.4 Member Data Documentation

6.117.4.1 size_t size_ [private]

The size.
Permitted size of the semaphore

6.117.4.2 std::mutex* mtx [private]

Locker.
Manages lock and unlock of the semaphore
The documentation for this class was generated from the following files:

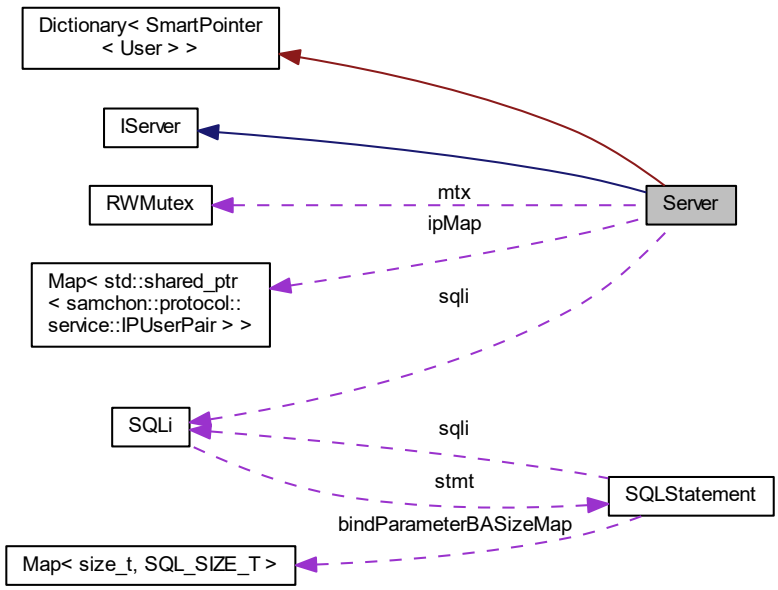
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Semaphore.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/Semaphore.cpp

6.118 Server Class Reference

A server for (cloud) service.

`#include <Server.hpp>`

Collaboration diagram for Server:



Public Member Functions

- Server ()
  Default Constructor.
- auto size () const -> size_t
  Size of User(s).
- auto begin () const -> const_iterator
  A const iterator of begin.
- auto end () const -> const_iterator
  A const iterator of end.

Protected Member Functions

- virtual auto NAME () const -> std::string=0
  A name can be an identifier of a cloud server.
- virtual auto createUser () -> User *=0
  Factory method of User.
- virtual void addClient (Socket *) override
  Handling connection of a client.

Protected Attributes

- library::SQLi * sqli
  SQLi for archiving log.

Private Attributes

- Dictionary< std::shared_ptr< IPUserPair > > ipMap

Dictionary of issuer of session ID of each ip.

* size_t sequence

  Sequence for issuing session ID.

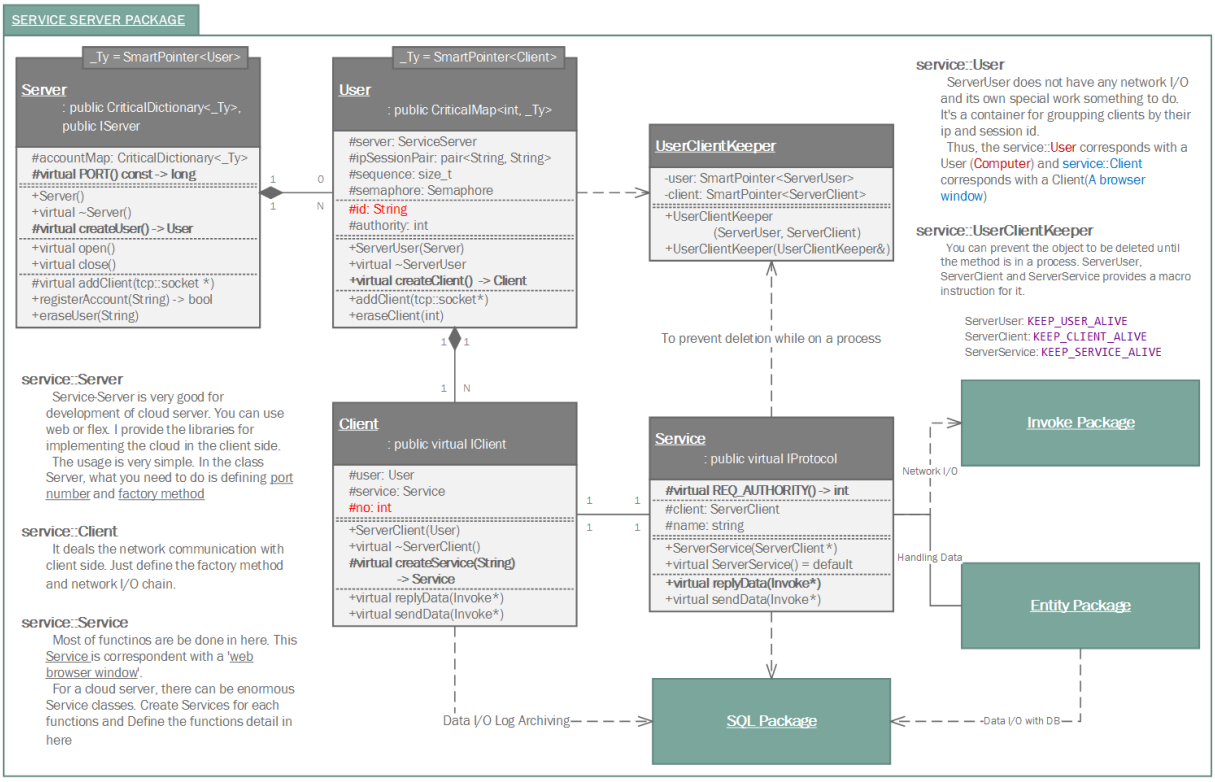
Additional Inherited Members

6.118.1 Detailed Description

A server for (cloud) service.

A server class representing the real physical server of a cloud.

The Server is very good for development of a cloud server. You can use web or flex as (physical) client. The usage is very simple. In the class Server, what you need to do are overriding getter of port number and factory method creating an User.



Note

Override those methods.

* Server::NAME()
* Server::PORT()
* Server::createUser()

See also

samchon::protocol
samchon::protocol::service

Author

Jeongho Nam

6.118.2 Member Function Documentation

6.118.2.1 auto begin ( ) const -> const_iterator

A const iterator of begin.

Note

Be careful about concurrency

6.118.2.2 auto end ( ) const -> const_iterator

A const iterator of end.

Note

Be careful about concurrency

The documentation for this class was generated from the following files:

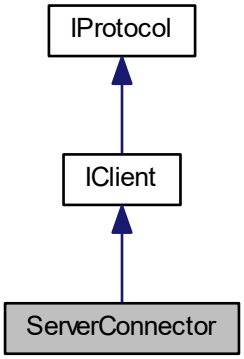
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/Server.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/Server.cpp

6.119 ServerConnector Class Reference

A server connector for a physical client.

`#include <ServerConnector.hpp>`

Collaboration diagram for ServerConnector:



Public Member Functions

* ServerConnector ()
  Default Constructor.
* virtual auto getIP () const -> std::string=0
  Destinatio IP.
* virtual auto getPort () const -> int=0
  Destination port.
* virtual auto getMyIP () const -> std::string
  (optional) My IP, if you want to bind
* virtual void connect ()
  Connect to a server.

Protected Attributes

* boost::asio::io_service * ioService
  An io_service of Boost.Asio's own.
* EndPoint * endPoint
  An endpoint directing a server.
* EndPoint * localEndPoint
  (Optional) An local endpoint of the client (my system)

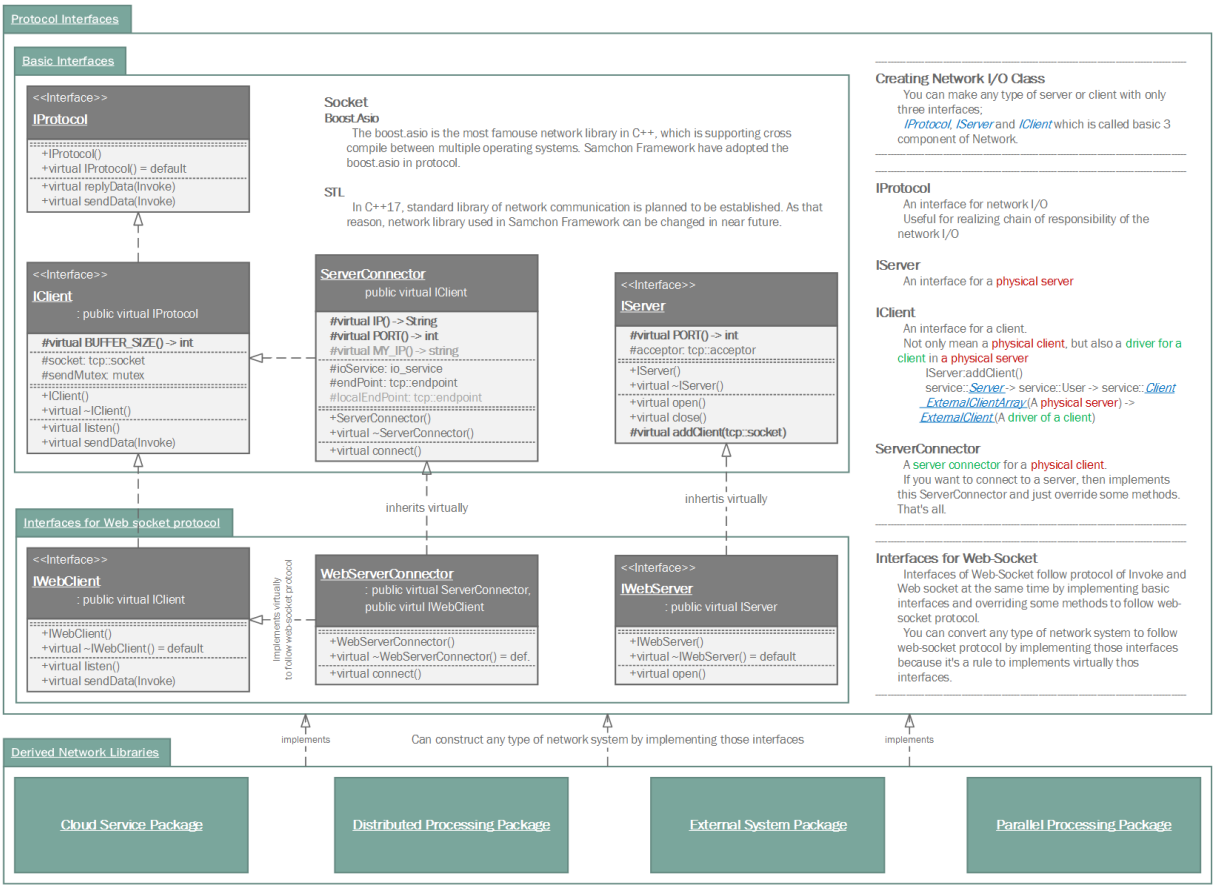
Additional Inherited Members

6.119.1 Detailed Description

A server connector for a physical client.

ServerConnector is a class for a physical client connecting a server. If you want to connect to a server, then implements this ServerConnector and just override some methods like getIP(), getPort() and replyData(). That's all.

In Samchon Framework, package protocol, There are basic 3 + 1 components that can make any type of network system in Samchon Framework. The basic 3 components are IProtocol, IServer and IClient. The last, surplus one is the ServerConnector. Looking around classes in Samchon Framework, especially module master and slave which are designed for realizing distributed processing systems and parallel processing systems, physical client classes are all derived from this ServerConnector.



Author

Jeongho Nam

6.119.2 Member Function Documentation

6.119.2.1 void connect ( ) [virtual]

Connect to a server.

Connects to a server with configured ip address and port number. After connection, listens data replied from the server (IClient::listen()).

Note

It monopolies a thread

Reimplemented in WebServerConnector.

6.119.3 Member Data Documentation

6.119.3.1 EndPoint* localEndPoint [protected]

(Optional) An local endpoint of the client (my system)

An local endpoint fof my system. It binds my ip to custom. It acts when my_ip is configured. If the my_ip is not configured, does not act.

The documentation for this class was generated from the following files:

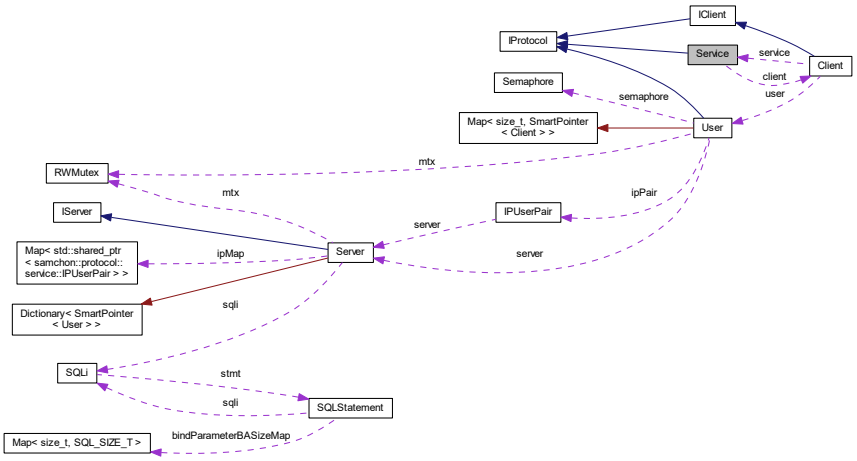
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ServerConnector.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/ServerConnector.cpp

6.120 Service Class Reference

A service.

`#include <Service.hpp>`

Collaboration diagram for Service:



Public Member Functions

* Service (Client *)
  Construct from Client.
* auto getClient () const -> Client *
  Get Client.
* virtual void sendData (std::shared_ptr< Invoke >)
  Shift the responsibility of sending an Invoke message to Client.

Protected Member Functions

* virtual auto REQUIRE_AUTHORITY () const -> int=NULL
  Required authority to access the service.

Private Attributes

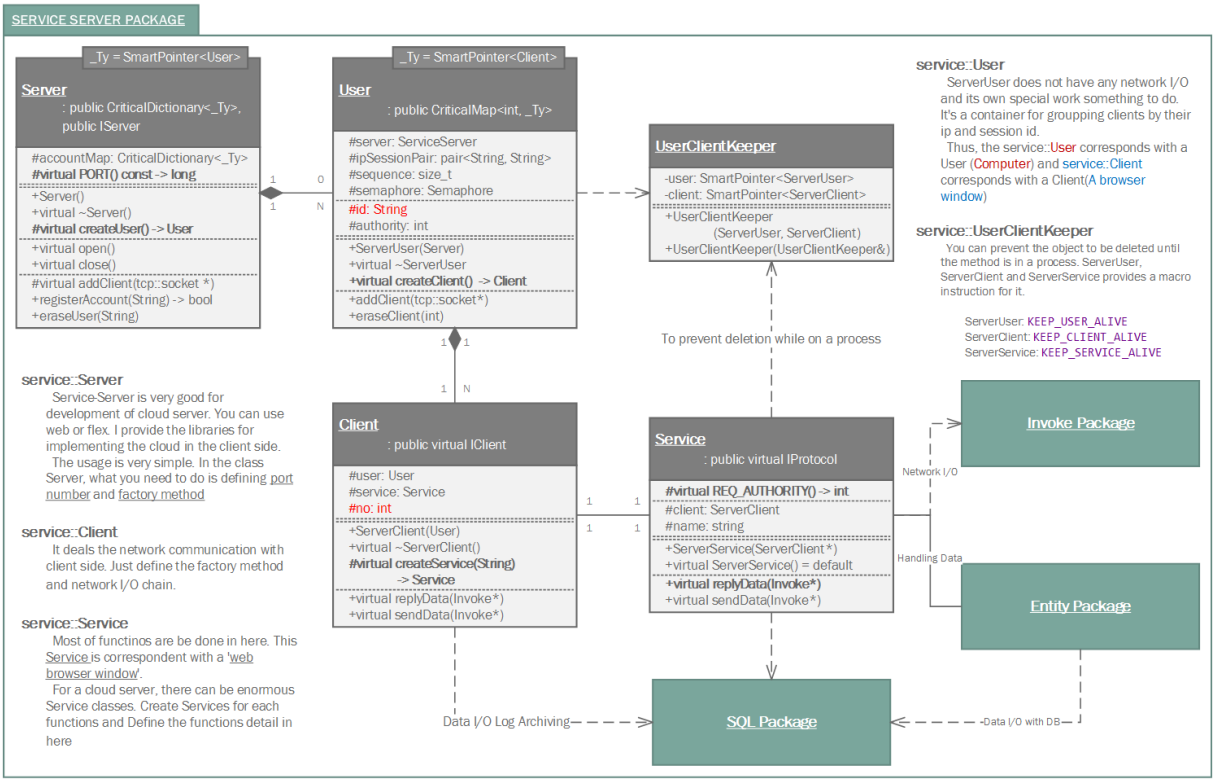
* Client * client
  Client object Service is belonged to.
* std::string name
  A name represents a Service.

6.120.1 Detailed Description

A service.

A service represents an unique functions of a web page

Most of functinos are realized in here. The Service is correspondent with a 'a service of a web page'. For a cloud Server, there can be enormous Service classes. Create Services for each functions and Define the functions detail in here.



See also

samchon::protocol
samchon::protocol::service

Author

Jeongho Nam

6.120.2 Constructor & Destructor Documentation

6.120.2.1 Service (Client * *client*)

Construct from Client.

Parameters

client	Client object Service is belonged to

6.120.3 Member Function Documentation

6.120.3.1 void sendData (std::shared_ptr< Invoke >) [virtual]

Shift the responsibility of sending an Invoke message to Client.

Parameters

in	An Invoke message to be sent to the (physical) client

Reimplemented from IProtocol.

The documentation for this class was generated from the following files:

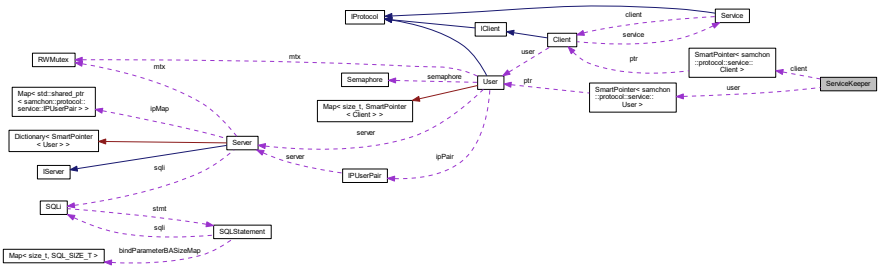
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/Service.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/Service.cpp

6.121 ServiceKeeper Class Reference

A keeper blocking destrunction of User and Client.

```
#include <ServiceKeeper.hpp>
```

Collaboration diagram for ServiceKeeper:



Public Member Functions

* ServiceKeeper (const User *user, const Client *client)
  Construct from User and Client.
* ServiceKeeper (const ServiceKeeper &)
  Copy Constructor.
* ServiceKeeper (ServiceKeeper &&)
  Movie Constructor.

Private Attributes

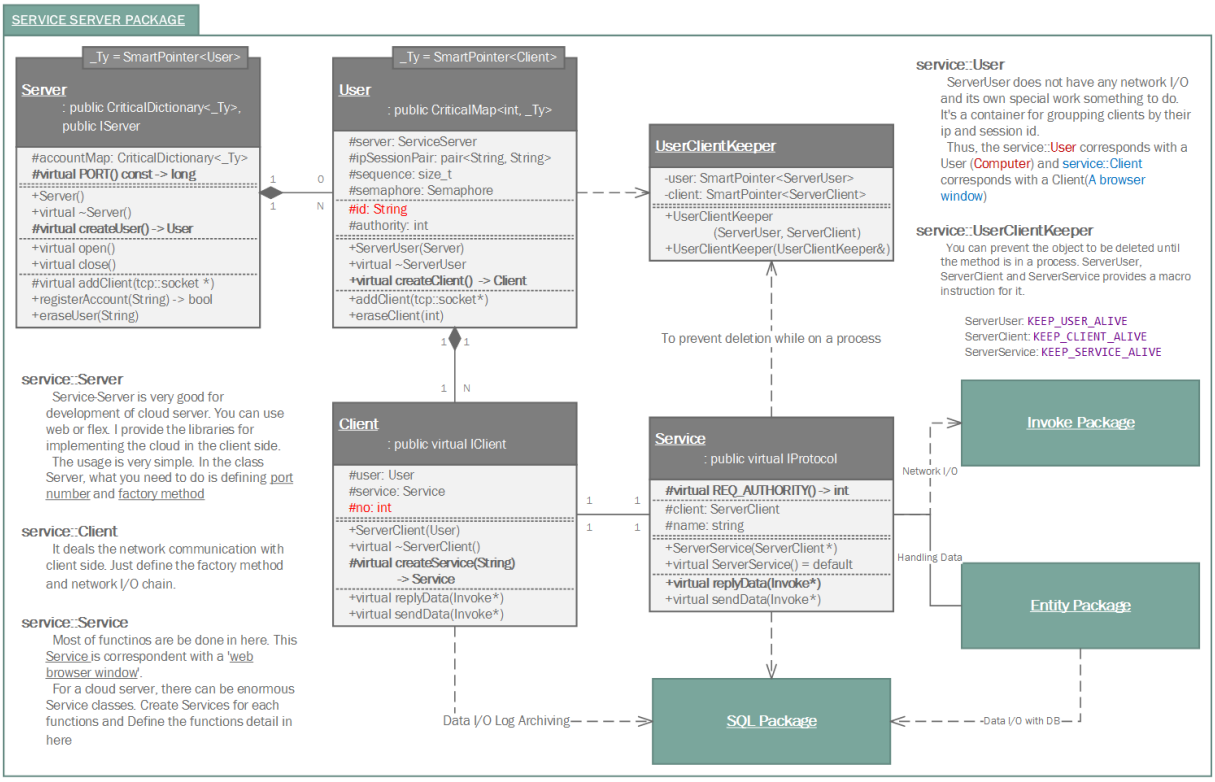
* SmartPointer< User > user
  A shared pionter of an User.
* SmartPointer< Client > client
  A shared pointer of a Client.

6.121.1 Detailed Description

A keeper blocking destrunction of User and Client.

Keeps User, Client and Service objects until termination of a stack.

You can prevent the object to be destructed until the method is in a process. User, Client and Service provides a macro instruction for it.



Note

Defined macros

- KEEP_USER_ALIVE
- KEEP_CLIENT_ALIVE
- KEEP_SERVICE_ALIVE

See also

samchon::protocol
samchon::protocol::service

Author

Jeongho Nam

6.121.2 Constructor & Destructor Documentation

6.121.2.1 ServiceKeeper (const User * *user*, const Client * *client*)

Construct from User and Client.

user An User to keep client A Client to keep

The documentation for this class was generated from the following files:

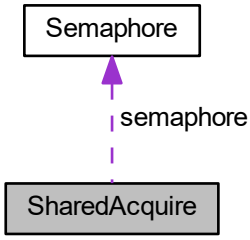
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/ServiceKeeper.hpp
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/ServiceKeeper.cpp

6.122 SharedAcquire Class Reference

Shared acquire from a Semaphore.

```
#include <SharedAcquire.hpp>
```

Collaboration diagram for SharedAcquire:



Public Member Functions

- SharedAcquire (Semaphore &, bool=true)
  Construct from Semaphore.
- SharedAcquire (const SharedAcquire &)
  Copy Constructor.
- SharedAcquire (SharedAcquire &&)
  Move Constructor.
- ~SharedAcquire ()
  Default Destructor.
- void acquire ()
  Acquire admission.
- void release ()
  Release an admission.

Private Attributes

- Semaphore * semaphore
  Managed Semaphore.
- std::atomic< size_t > * reference
  Referencing count sharing same Semaphore.
- std::atomic< bool > * isLocked
  Whether the mutex was locked by SharedAcquire.

6.122.1 Detailed Description

Shared acquire from a Semaphore.

A SharedAcquire is an object manages a Semaphore with shared ownership in both states.

- acquired
- released

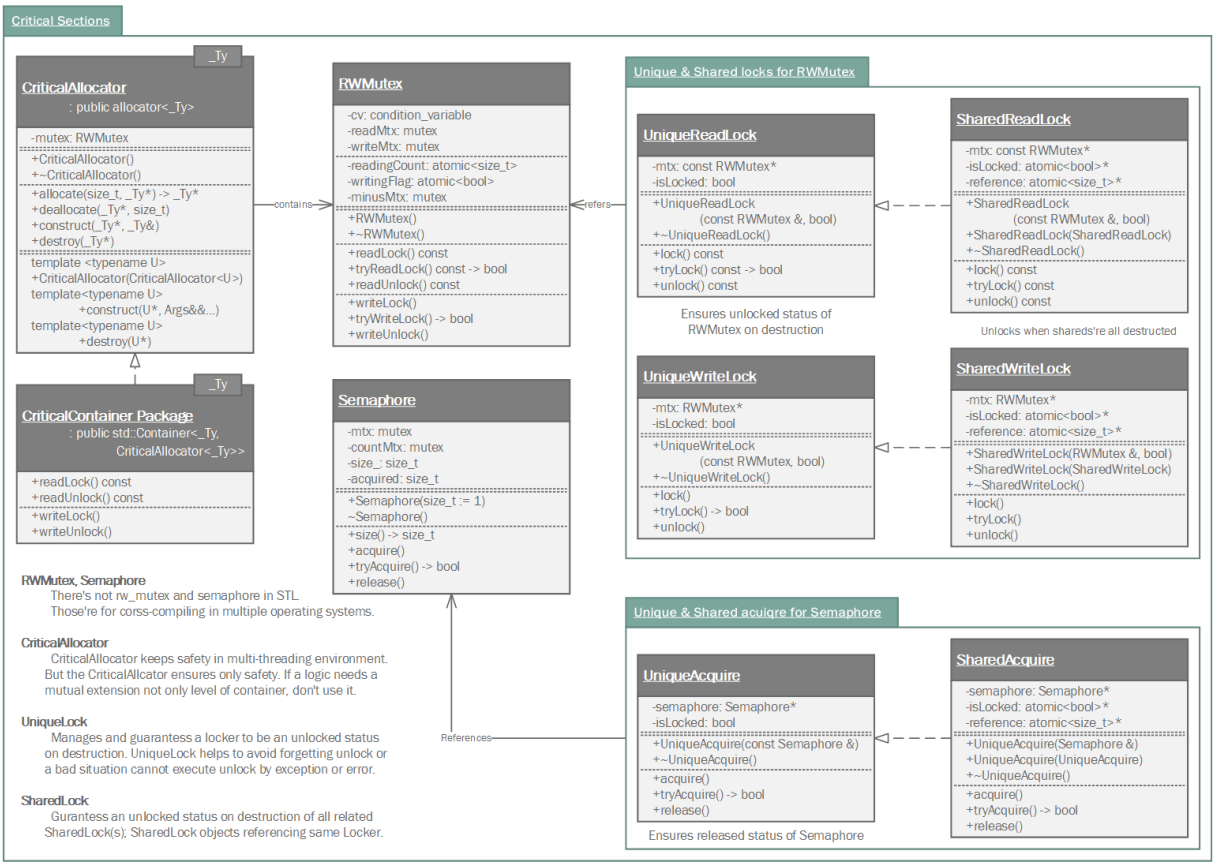
On default construction, the object acquires a Semaphore object, for whose acquiring and releasing operations becomes responsible. When copy constructions, responsibilities of acquiring and releasing Semaphore are shared with copied with those SharedAcquire objects.

The class shared_lock is a general-purpose shared mutex ownership wrapper allowing deferred locking, timed locking and transfer of lock ownership. Locking a shared_lock locks the associated shared mutex in shared mode (to lock it in exclusive mode, std::unique_lock can be used)

This class guarantees a released status on destruction of all shared objects (even if not called explicitly). Therefore it is especially useful as an object with automatic duration, as it guarantees the Semaphore object is properly released in case an exception is thrown.

Referenced comments of std::unique_lock

- http://www.cplusplus.com/reference/mutex/unique_lock/



Note

Though, that the SharedAcquire object does not manage the lifetime of the Semaphore object in any way: the duration of the Semaphore object shall extend at least until the destruction of the Shared↩ Acquire that manages it.

See also

library::UniqueAcquire
samchon::library

Author

Jeongho Nam

6.122.2 Constructor & Destructor Documentation

6.122.2.1 SharedAcquire (Semaphore & *semaphore*, bool *doLock* = `true`)

Construct from Semaphore.

Parameters

semaphore	Semaphore to manage
doLock	Whether to lock directly or not

6.122.3 Member Function Documentation

6.122.3.1 void acquire ( )

Acquire admission.

Acquires an admission and increases count of admission by 1.

If the count is over permitted size, wait until other admissions to be released.

* Lock on mutex

6.122.3.2 void release ( )

Release an admission.

Releases an admission what you've acquired. If the admission count was over the limited size, unlock the mutex.

The documentation for this class was generated from the following files:

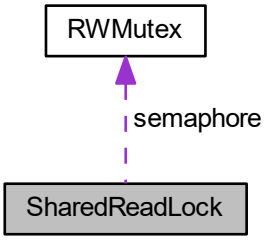
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SharedAcquire.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SharedAcquire.cpp

6.123 SharedReadLock Class Reference

Shared lock from a RWMutex.

`#include <SharedReadLock.hpp>`

Collaboration diagram for SharedReadLock:



Public Member Functions

* SharedReadLock (const RWMutex &, bool=true)
  Construct from RWMutex.
* SharedReadLock (const SharedReadLock &)
  Copy Constructor.
* SharedReadLock (SharedReadLock &&)
  Move Constructor.
* ~SharedReadLock ()
  Default Destructor.
* void lock () const
  Lock on read.
* void unlock () const
  Unlock of read.

Private Attributes

* const RWMutex * semaphore
  Managed RWMutex.
* std::atomic< size_t > * reference
  Referencing count sharing same RWMutex.
* std::atomic< bool > * isLocked
  Whether the mutex was locked by SharedReadLock.

6.123.1 Detailed Description

Shared lock from a RWMutex.

A SharedReadLock is an object manages a RWMutex with shared ownership in both states.

* locked
* unlocked

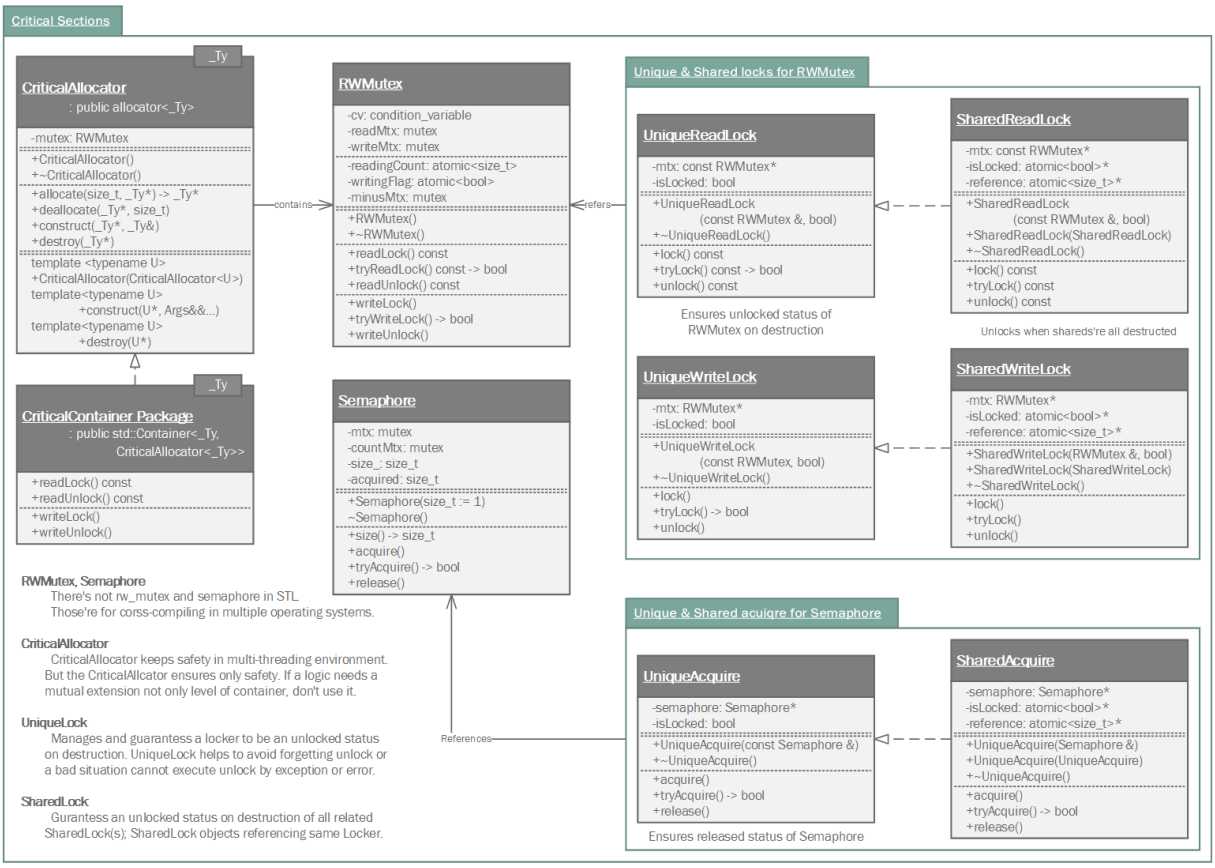
On default construction, the object acquires a RWMutex object, for whose locking and unlocking operations becomes responsible. When copy constructions, responsibilities of locking and unlocking RW↩ Mutex are shared with copied with those SharedReadLock objects.

The class shared_lock is a general-purpose shared mutex ownership wrapper allowing deferred locking, timed locking and transfer of lock ownership. Locking a shared_lock locks the associated shared mutex in shared mode (to lock it in exclusive mode, std::unique_lock can be used)

This class guarantees a unlocked status on destruction of all shared objects (even if not called explicitly). Therefore it is especially useful as an object with automatic duration, as it guarantees the RWMutex object is properly unlocked in case an exception is thrown.

Referenced comments of std::unique_lock

* http://www.cplusplus.com/reference/mutex/unique_lock/



Note

Though, that the SharedReadLock object does not manage the lifetime of the RWMutex object in any way: the duration of the RWMutex object shall extend at least until the destruction of the SharedRead↩ Lock that manages it.

See also

library::UniqueAcquire
samchon::library

Author

Jeongho Nam

6.123.2 Constructor & Destructor Documentation

6.123.2.1 SharedReadLock (const RWMutex & *semaphore*, bool *doLock* = `true`)

Construct from RWMutex.

Parameters

semaphore	RWMutex to manage
doLock	Whether to lock directly or not

6.123.3 Member Function Documentation

6.123.3.1 void lock ( ) const

Lock on read.

Increases a reading count.

When write_lock is on a progress, wait until write_unlock to be called.

* Reading can be done by multiple sections.
* Reading can't be done when writing.

Warning

You've to call read_unlock when the reading work is terminated.

6.123.3.2 void unlock ( ) const

Unlock of read.

Decreases a reading count.

When write_lock had done after read_lock, it continues by read_unlock if the reading count was 1 (read_unlock makes the count to be zero).

The documentation for this class was generated from the following files:

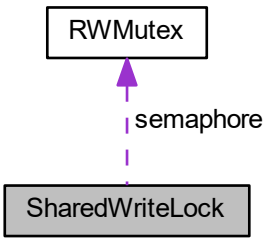
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SharedReadLock.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SharedReadLock.cpp

6.124 SharedWriteLock Class Reference

Shared lock from a RWMutex.

`#include <SharedWriteLock.hpp>`

Collaboration diagram for SharedWriteLock:



Public Member Functions

* SharedWriteLock (RWMutex &, bool=true)
  Construct from RWMutex.
* SharedWriteLock (const SharedWriteLock &)
  Copy Constructor.

* SharedWriteLock (SharedWriteLock &&)
  Move Constructor.
* ~SharedWriteLock ()
  Default Destructor.
* void lock ()
  Lock on writing.
* void unlock ()
  Unlock on writing.

Private Attributes

* RWMutex * semaphore
  Managed RWMutex.
* std::atomic< size_t > * reference
  Referencing count sharing same RWMutex.
* std::atomic< bool > * isLocked
  Whether the mutex was locked by SharedWriteLock.

6.124.1 Detailed Description

Shared lock from a RWMutex.

A SharedWriteLock is an object manages a RWMutex with shared ownership in both states.

* locked
* unlocked

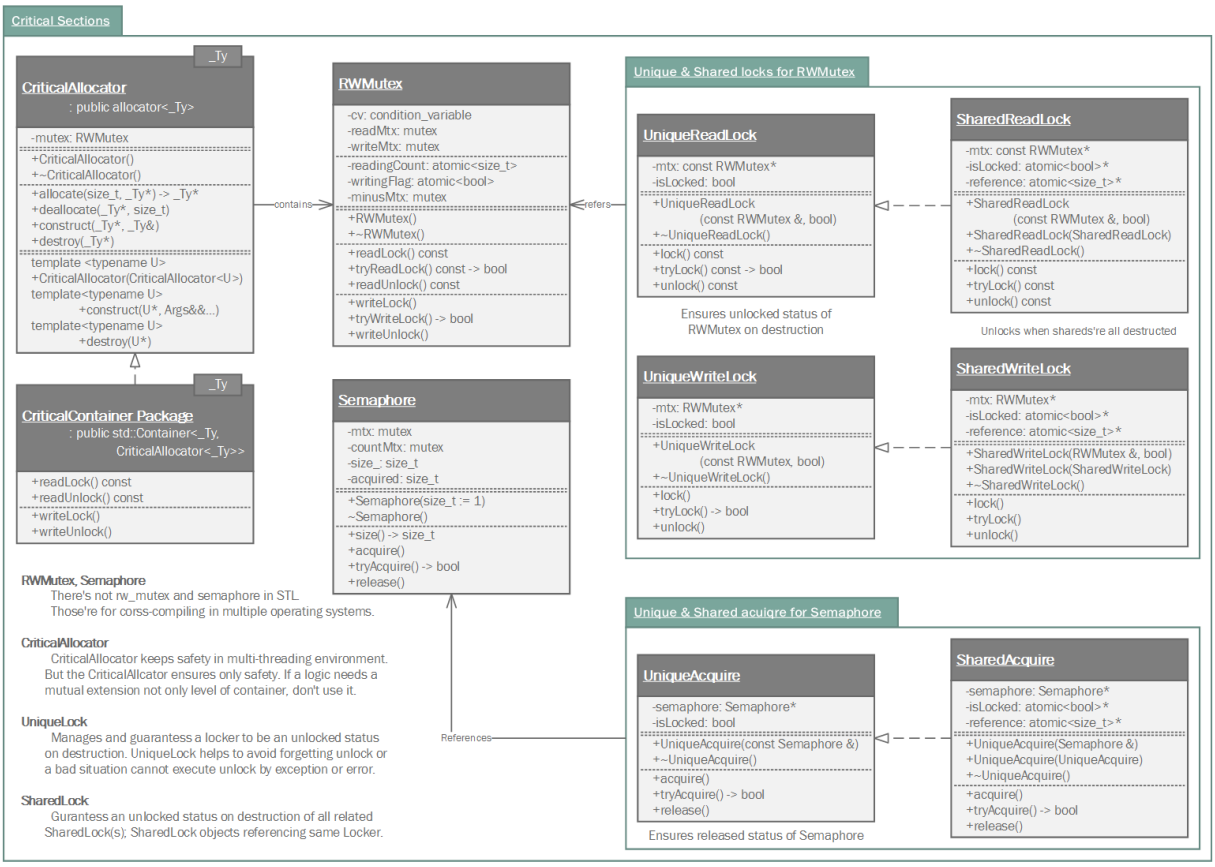
On default construction, the object acquires a RWMutex object, for whose locking and unlocking operations becomes responsible. When copy constructions, responsibilities of locking and unlocking RWMutex are shared with copied with those SharedWriteLock objects.

The class shared_lock is a general-purpose shared mutex ownership wrapper allowing deferred locking, timed locking and transfer of lock ownership. Locking a shared_lock locks the associated shared mutex in shared mode (to lock it in exclusive mode, std::unique_lock can be used)

This class guarantees a unlocked status on destruction of all shared objects (even if not called explicitly). Therefore it is especially useful as an object with automatic duration, as it guarantees the RWMutex object is properly unlocked in case an exception is thrown.

Referenced comments of std::unique_lock

* http://www.cplusplus.com/reference/mutex/unique_lock/



Note

Though, that the SharedWriteLock object does not manage the lifetime of the RWMutex object in any way: the duration of the RWMutex object shall extend at least until the destruction of the SharedWrite↩Lock that manages it.

See also

library::UniqueAcquire
samchon::library

Author

Jeongho Nam

6.124.2 Constructor & Destructor Documentation

6.124.2.1 SharedWriteLock (RWMutex & *semaphore*, bool *doLock* = `true`)

Construct from RWMutex.

Parameters

semaphore	RWMutex to manage
doLock	Whether to lock directly or not

6.124.3 Member Function Documentation

6.124.3.1 void lock ( )

Lock on writing.

Changes writing flag to true.

If another write_lock or read_lock is on a progress, wait until them to be unlocked.

* Writing can be done by only a section at once.
* Writing can't be done when reading.

Note

You've to call write_unlock when writing work was terminated.

6.124.3.2 void unlock ( )

Unlock on writing.

The documentation for this class was generated from the following files:

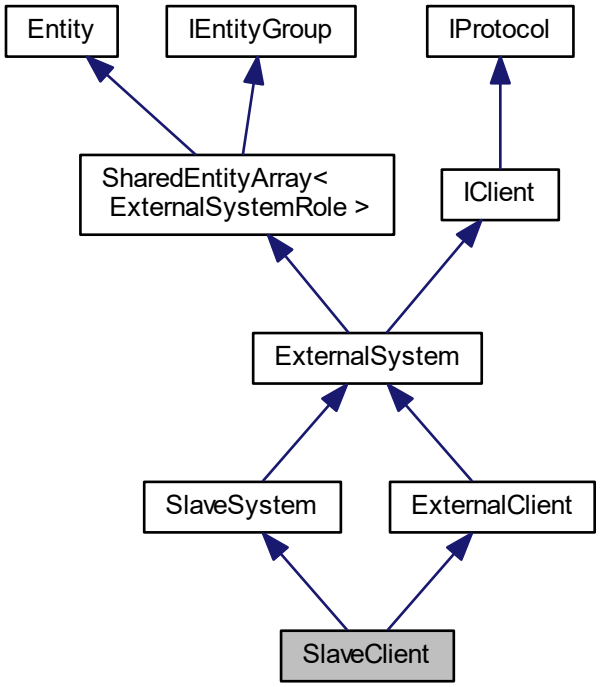
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SharedWriteLock.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SharedWriteLock.cpp

6.125 SlaveClient Class Reference

A slave client.

`#include <SlaveClient.hpp>`

Collaboration diagram for SlaveClient:



Public Member Functions

* SlaveClient ()
  Default Constructor.

Additional Inherited Members

6.125.1 Detailed Description

A slave client.

SlaveClient is a SlaveSystem specialized in client driver.

[Inherited]

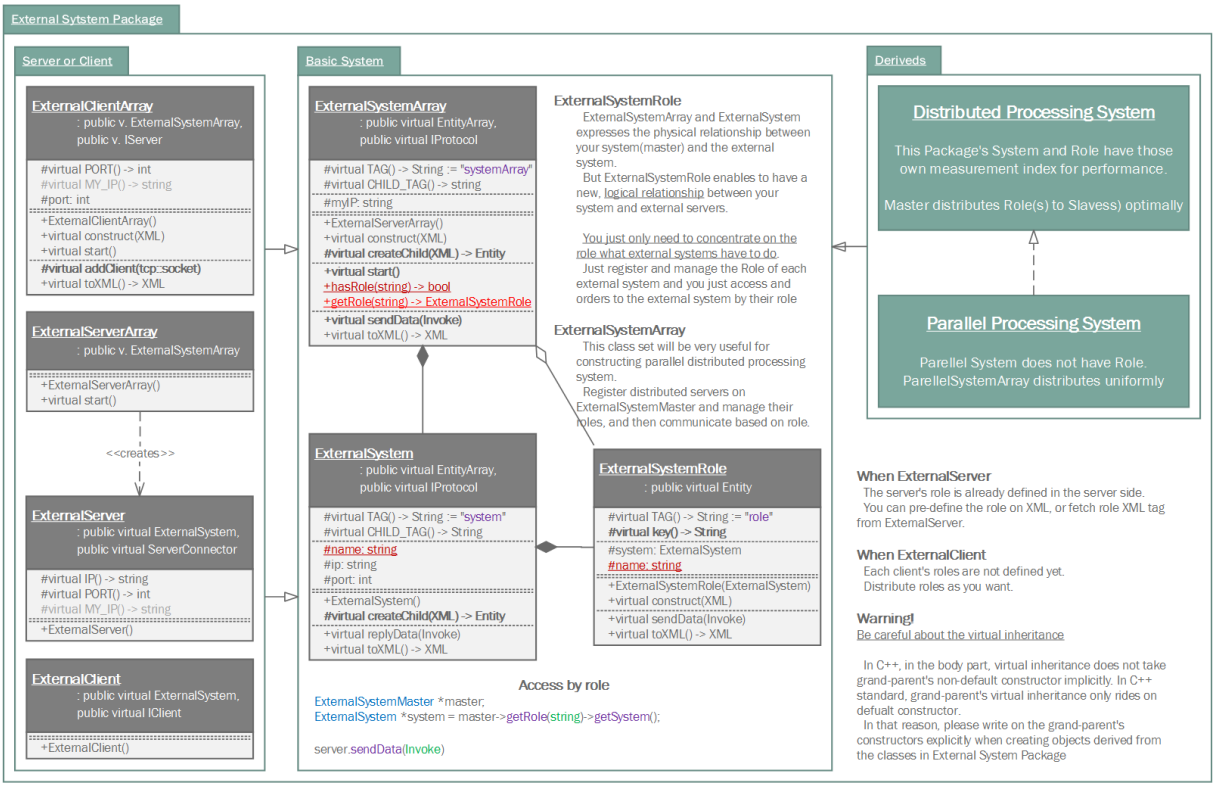
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::↩ ParallelSystem and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem,

and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

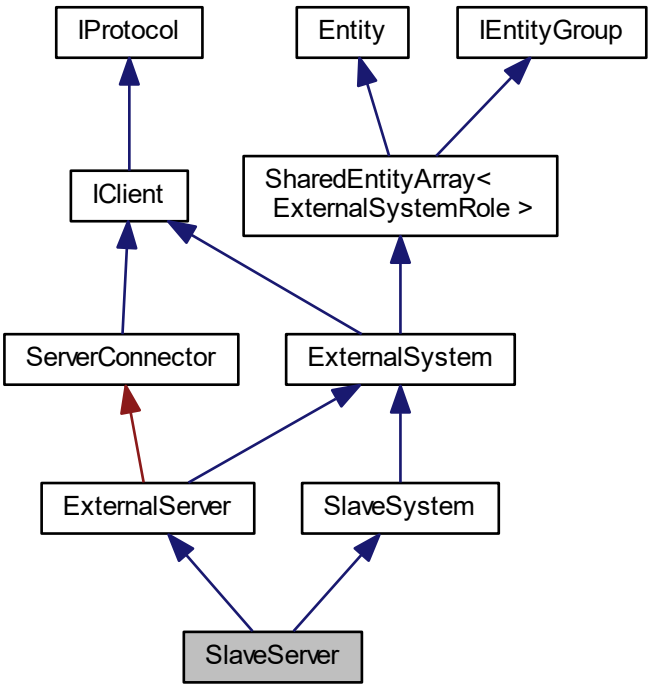
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/slave/SlaveClient.hpp

6.126 SlaveServer Class Reference

A slave server.

`#include <SlaveServer.hpp>`

Collaboration diagram for SlaveServer:



Public Member Functions

* SlaveServer ()
  Default Constructor.

Additional Inherited Members

6.126.1 Detailed Description

A slave server.

SlaveServer is a SlaveSystem specialized in server driver.

[Inherited]

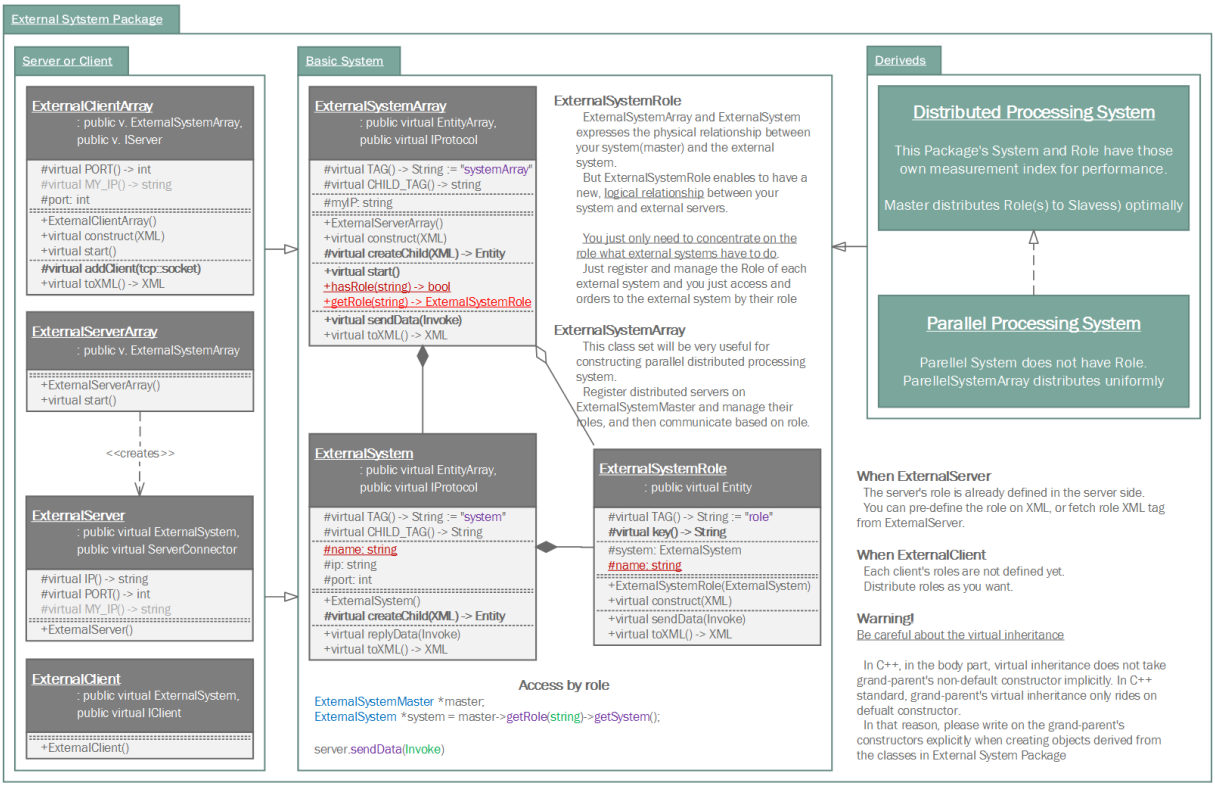
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::↩ ParallelSystem and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem,

and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

The documentation for this class was generated from the following file:

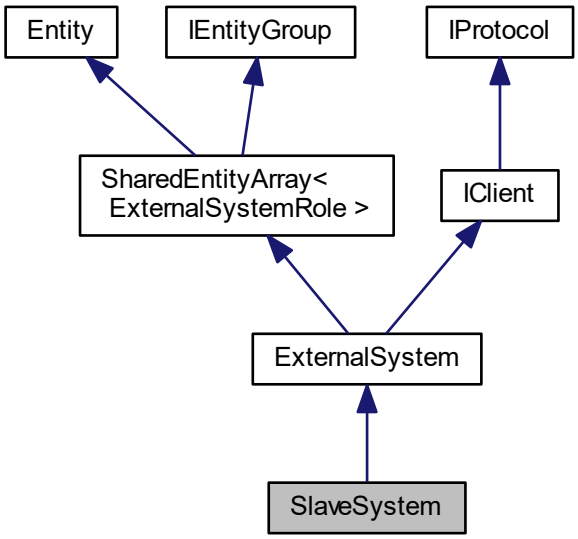
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/slave/SlaveServer.hpp

6.127 SlaveSystem Class Reference

A slave system.

`#include <SlaveSystem.hpp>`

Collaboration diagram for SlaveSystem:



Public Member Functions

* SlaveSystem ()
  Default Constructor.

Protected Member Functions

* virtual void _replyData (std::shared_ptr< Invoke >) override
  Pre-processor for reporting elapsed time.

Additional Inherited Members

6.127.1 Detailed Description

A slave system.

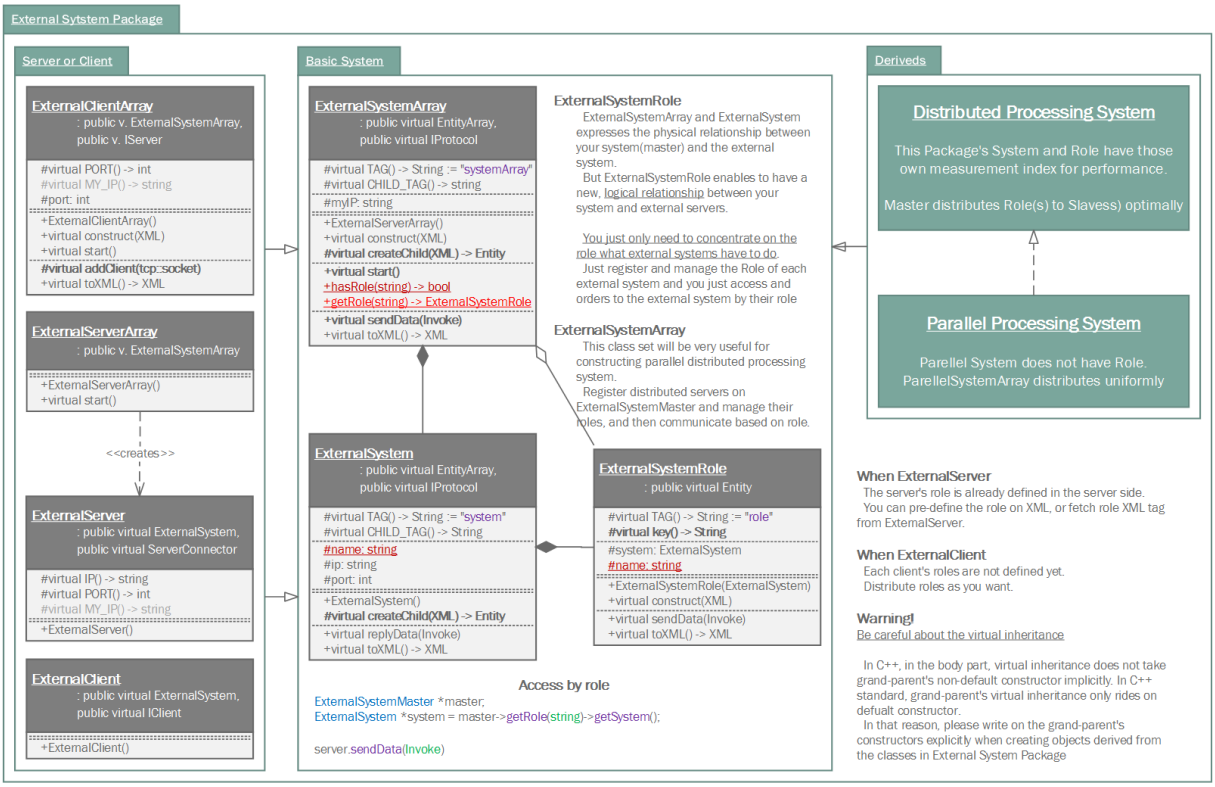
SlaveSystem, literally, means a slave system belongs to a maste system.

The SlaveSystem class is used in opposite side system of master::DistributedSystem and master::↩ ParallelSystem and reports elapsed time of each commmand (by Invoke message) for estimation of its performance.

[Inherited]

ExternalSystem is a boundary class interacting with an external system by network communication. Also, ExternalSystem is an abstract class that a network role, which one is server and which one is client, is not determined yet.

The ExternalSystem has ExternalSystemRole(s) groupped methods, handling Invoke message interacting with the external system, by subject or unit of a moudle. The ExternalSystemRole is categorized in a 'control'.



Note

The ExternalSystem class takes a role of interaction with external system in network level. However, within a framework of Samchon Framework, a boundary class like the ExternalSystem is not such important. You can find some evidence in a relationship between ExternalSystemArray, ExternalSystem and ExternalSystemRole.

Of course, the ExternalSystemRole is belonged to an ExternalSystem. However, if you access an ExternalSystemRole from an ExternalSystemArray directly, not passing by a belonged ExternalSystem, and send an Invoke message even you're not knowing which ExternalSystem is related in, it's called "Proxy pattern".

Like the explanation of "Proxy pattern", you can utilize an ExternalSystemRole as a proxy of an ExternalSystem. With the pattern, you can only concentrate on ExternalSystemRole itself, what to do with Invoke message, irrespective of the ExternalSystemRole is belonged to which ExternalSystem.

See also

samchon::protocol

Author

Jeongho Nam

6.127.2 Member Function Documentation

6.127.2.1 virtual void _replyData (std::shared_ptr< Invoke >) [override], [protected], [virtual]

Pre-processor for reporting elapsed time.

[Inherited]

You can realize a pre-processing about the replied Invoke message by overriding the IClient::_reply↩Data().

Archiving a history log, filtering and truncating, let handling the message with a new thread, calculating performance by estimating elapsed time about the Invoke message, and so on. Anything you can define pre-processing works before IProtocol::replyData().

* Archive history log -> cloud service
* Estimating performance -> distributed and parallel processing system

Parameters

invoke	An invoke message to be pre-processed and shifted to IProtocol::replyData().

Reimplemented from IClient.

Reimplemented in ParallelSystem.

The documentation for this class was generated from the following file:

* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/slave/SlaveSystem.hpp

6.128 SmartPointer< _Ty > Class Template Reference

Global shared pointer

.

```
#include <SmartPointer.hpp>
```

Public Member Functions

* SmartPointer ()
  Default Constructor.
* SmartPointer (const _Ty *ptr)
  Constrct from pointer.
* SmartPointer (const SmartPointer &smartPointer)
  Copy Constructor.
* SmartPointer (SmartPointer &&smartPointer)
  Move constructor.
* ~SmartPointer ()
  Destroy SmartPointer.
* void reset (const _Ty *ptr)
  Reset pointer.
* auto get () const -> _Ty *
  Get pointer.
* auto operator-> () const -> _Ty *
  Dereference object membr.
* auto operator* () const -> _Ty &
  Dereference object.

Private Attributes

* _Ty * ptr
  A pointer managed by SmartPointer.

Static Private Attributes

* static std::map< _Ty *, size_t > useCountMap
  Map of use count of each pointer.
* static std::mutex mtx
  Mutex assigned to useCountMap.

6.128.1 Detailed Description

template<typename _Ty>class samchon::SmartPointer< _Ty >

Global shared pointer

.

SmartPointer is a global shared pointer based on map, map referencing.

Manages the storage of a pointer, providing a limited garbage-collection facility, possibly sharing that management with other objects.

Referenced comments of std::allocator.

* http://www.cplusplus.com/reference/memory/shared_ptr/

Template Parameters

The	type of managed object

Author

Jeongho Nam

6.128.2 Constructor & Destructor Documentation

6.128.2.1 SmartPointer ( ) [inline]

Default Constructor.

The object is empty (owns no pointer, use count of zero)

6.128.2.2 SmartPointer (const _Ty * *ptr*) [inline],[explicit]

Constrct from pointer.

The object owns ptr, setting the use count to add 1

Parameters

ptr	The pointer to own

6.128.2.3 SmartPointer (const SmartPointer< _Ty > & *smartPointer*) [inline]

Copy Constructor.

The object shares ownership of smartPointer's asset and increases the use count

Parameters

smartPointer	The object to copy

6.128.2.4 SmartPointer (SmartPointer< _Ty > && *smartPointer*) [inline]

Move constructor.

The object acquires the content managed by smartPointer

The ceding object becomes empty and there's no change on use count

Parameters

smartPointer	The object to move

6.128.2.5 ~SmartPointer ( ) [inline]

Destroy SmartPointer.

Destroys the object. But, before, it may produce the following side effects depending on the use_count of member

- If use_count is greater than 1: The use count is decreased by 1.
- If use_count is 1 (i.e., the object is the unique owner of the managed pointer): the object pointed by its owned pointer is deleted.
- If use_count is zero (i.e., the object is empty), this destructor has no side effects.

6.128.3 Member Function Documentation

6.128.3.1 void reset (const _Ty * *ptr*) [inline]

Reset pointer.

Reset pointer to manage and shrink use count of previous pointer.

Additionally, a call to this function has the same side effects as if SmartPointer's destructor was called before its value changed (including the deletion of the managed object if this SmartPointer was unique).

Parameters

Pointer	whose ownership is taken over by the object. Unlike std::shared_ptr, ptr being managed by another SmartPointer does not cause any problem

6.128.3.2 auto get ( ) const -> _Ty* [inline]

Get pointer.

Returns the stored pointer.

The stored pointer points to the object the shared_ptr object dereferences to, which is generally the same as its owned pointer.

Returns

The stored pointer

6.128.3.3 auto operator-> ( ) const -> _Ty* [inline]

Dereference object membr.

Returns a pointer to the object pointed by the stored pointer in order to access one of its members. This member function shall not be called if the stored pointer is a null pointer.

It returns the same value as get().

Returns

A pointer to be managed by SmartPointer

6.128.3.4 auto operator* ( ) const -> _Ty& [inline]

Dereference object.

Returns a reference to the object pointerd by pointer. It is equivalent to: *get()

Returns

A reference to the object pointed

The documentation for this class was generated from the following file:

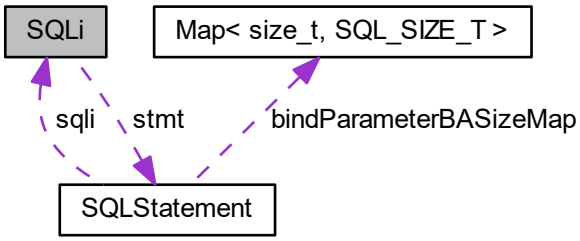
- D:/OneDrive/Project/Samchon/framework/cpp/samchon/SmartPointer.hpp

6.129 SQLi Class Reference

A SQL interface; DBMS connector.

`#include <SQLi.hpp>`

Collaboration diagram for SQLi:



Public Member Functions

- SQLi (const std::string &driver, int port)
  Construct from driver name and port.
- virtual void connect (const std::string &ip, const std::string &db, const std::string &id, const std↩ ::string &pwd)
  Connect to the DBMS
  .
- virtual void disconnect ()
  Disconnect from DBMS.
- virtual auto createStatement () -> std::shared_ptr< SQLStatement >
  Factory method for creating SQLStatement.

Protected Attributes

- std::string driver
  Driver; name of DBMS used for connection.
- int port
  Port number of DBMS.

Private Attributes

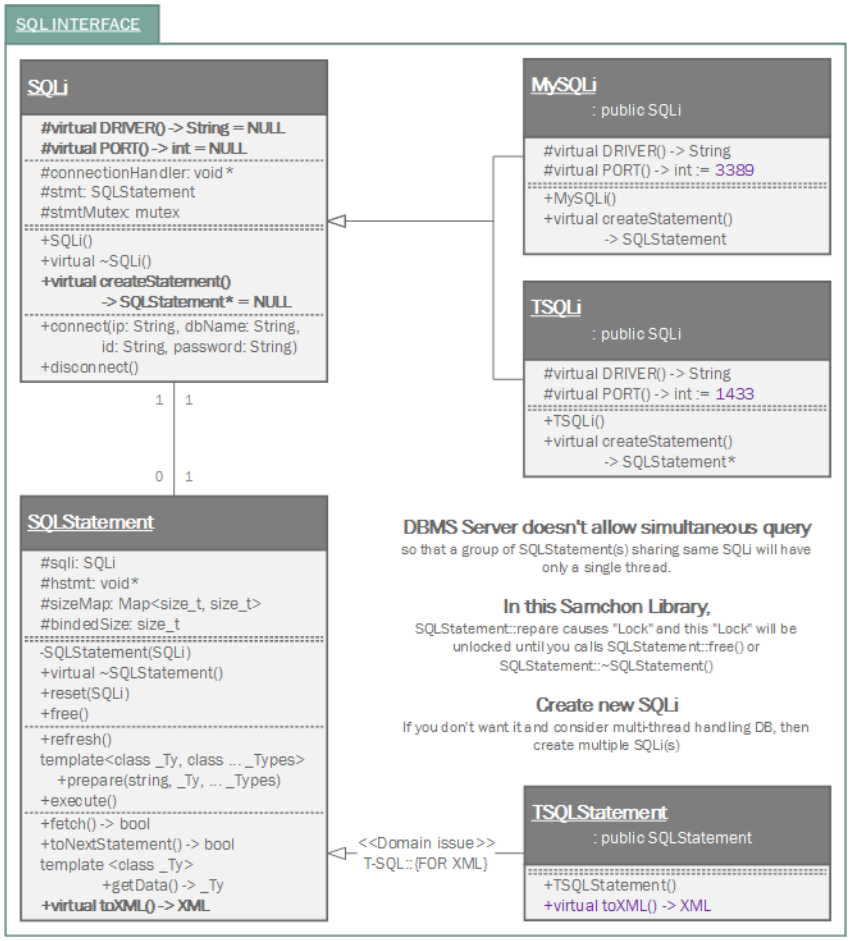
- void * hdbc
  Handler of DB-connector.
- SQLStatement * stmt
  SQLStatement's pointer linked with the SQLi.
- std::mutex stmtMutex
  Mutex ensuring concurrency with SQLStatement.

6.129.1 Detailed Description

A SQL interface; DBMS connector.

SQLi is an abstract class managing connection (session) with a specific database and manages SQL statements are executed and results are returned within the context of a connection.

The reason why SQLStatement(s) are managed by SQLi is because of a principle of DBMS, DBM↩ S system doesn't allow simultaneous query from a session (connection). Only a query (process0) is allowed at a time. If you try to simultaneous query from a SQLi, ODBC throws error. To avoid the error, an execute of query from SQLStatement will lock a mutex of SQLi to ensure exclusiveness.



Note

To ensure the exclusiveness, you've to make SQLStatement from SQLi. Do not make SQLStatement by yourself. call SQLi::createSQLStatement() instead. Even you make a derived class from SQLStatement, don't make its constructor to have public accessor.

Becuase execution of a sql statement causes lock on mutex in SQLi, you've to destruct the SQL↩ Statement or call SQLStatement::free() method(). If you don't, the mutex will not be unlocked, thus you can't do anything by the SQLi.

See also

samchon::library

Author

Jeongho Nam

6.129.2 Constructor & Destructor Documentation

6.129.2.1 SQLi (const std::string & *driver*, int *port*)

Construct from driver name and port.

Parameters

driver	Driver name of DBMS

port	Port number of DBMS

6.129.3 Member Function Documentation

6.129.3.1 void connect (const std::string & *ip*, const std::string & *db*, const std::string & *id*, const std::string & *pwd*) [virtual]

Connect to the DBMS

.

Parameters

ip	An IP address of the DBMS
db	A name of database to access
id	Your own ID authorified from DBMS
pwd	A password for the ID

Exceptions

exception	Error message from DBMS

6.129.3.2 auto createStatement ( ) -> std::shared_ptr<SQLStatement> [virtual]

Factory method for creating SQLStatement.

Recommend to create SQLStatement by this method. Direct creation is not recommended as the reason of domain problem of each DBMS

Returns

A SQLStatement matched for the domain SQLi

Reimplemented in TSQLi.

6.129.4 Member Data Documentation

6.129.4.1 std::string driver [protected]

Driver; name of DBMS used for connection.

* MySQL: "{MySQL Server}"
* MS-SQL: "{SQL Server}"
* PL-SQL: "{Oracle}"

6.129.4.2 int port [protected]

Port number of DBMS.

* MySQL: 3306
* MS-SQL: 1433
* PL-SQL: 1521

The documentation for this class was generated from the following files:

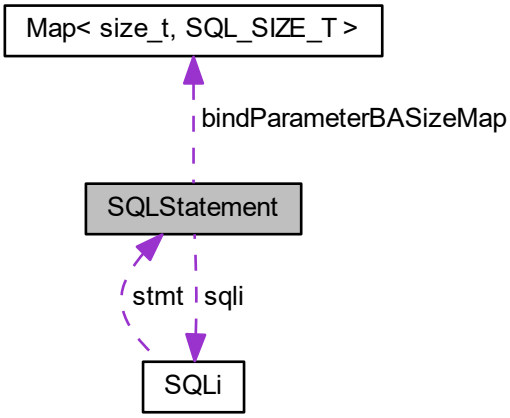
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SQLi.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SQLi.cpp

6.130 SQLStatement Class Reference

A sql statement.

```
#include <SQLStatement.hpp>
```

Collaboration diagram for SQLStatement:



Public Member Functions

- void free ()
  Free the sql statement.
- void refresh ()
  Refresh the sql statement.
- template<typename _Ty , typename... _Args>
  void prepare (const std::string &sql, const _Ty &val, const _Args &...args)
  Prepare a sql statement.
- void execute ()
  Execute the prepared sql statement.
- void executeDirectly (const std::string &)
  Execute sql-statement direclty.
- auto fetch () const -> bool
  Fetch a record.
- auto next () const -> bool
  Move cursor to the next sql-statement.
- auto size () const -> size_t
  Get size of columns.
- template<typename _Ty >
  auto at (size_t index) const -> _Ty
  Get column's data by its index.
- template<typename _Ty >
  auto get (const std::string &) const -> _Ty
  Get a column data by its name.
- virtual auto toXML () const -> std::shared_ptr< XML >
  Result sets to XML.

Protected Member Functions

- SQLStatement (SQLi *sqli)
  Protected Constructor.

Protected Attributes

- SQLi * sqli
  SQLi who created the SQLStatement.
- void * hstmt
  Handler of sql statement (OBDC)

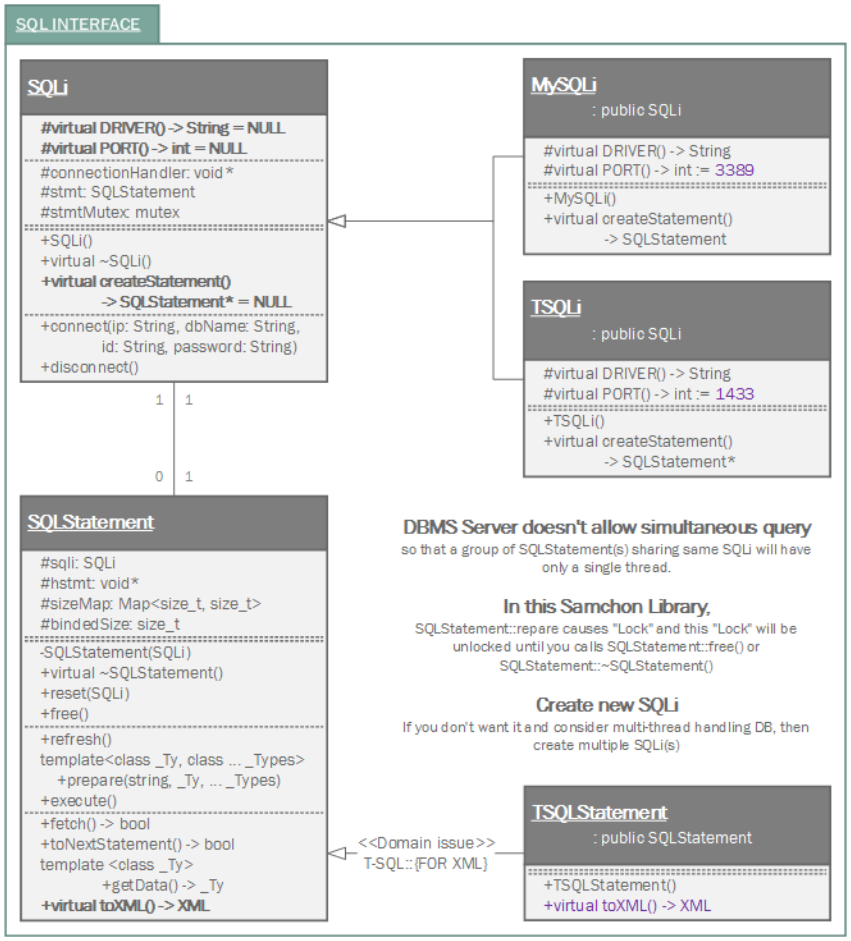
* size_t bindParameterCount
  Count of binded parameters
  .
* Map< size_t, SQL_SIZE_T > bindParameterBASizeMap
  A map for binary size.

6.130.1 Detailed Description

A sql statement.

A SQLStatement instance is used to executing a SQL statement and returning the results it produces against a SQL database that is opened through a SQLi instance.

Through the SQLi, it's the reason why a principle of DBMS, DBMS system doesn't allow simultaneous query from a session (connection). Only a query (process) is allowed at a time. If you try to simultaneous query from a SQLi, ODBC throws error. To avoid the error, an execute of query from SQLStatement will lock a mutex of SQLi to ensure exclusiveness.



Note

To ensure the exclusiveness, you've to make SQLStatement from SQLi. Do not make SQLStatement by yourself. call SQLi::createSQLStatement() instead. Even you make a derived class from SQLStatement, don't make its constructor to have public accessor.

Becuase execution of a sql statement causes lock on mutex in SQLi, you've to destruct the SQL↩ Statement or call SQLStatement::free() method(). If you don't, the mutex will not be unlocked, thus you can't do anything by the SQLi.

See also

samchon::library

Author

Jeongho Nam

6.130.2 Constructor & Destructor Documentation

6.130.2.1 SQLStatement (SQLi * *sqli*) [protected]

Protected Constructor.

SQLStatement's constructor have to created by SQLi::createStatement().

Note

Don't create SQLStatement by yourself.

SQLStatement has to be created by SQLi::createStatement().

Parameters

sqli	Parent SQLi who created the SQLStatement

6.130.3 Member Function Documentation

6.130.3.1 void prepare (const std::string & *sql*, const _Ty & *val*, const _Args &... *args*) [inline]

Prepare a sql statement.

Prepare a sql statement with parameters to bind for execution

Warning

Be careful for destructions of binded parameters

Parameters

sql	A sql-statement to prepare
...	args The parameters to bind

6.130.3.2 void execute ( )

Execute the prepared sql statement.

Exceptions

exception	Error message from DBMS

6.130.3.3 void executeDirectly (const std::string & *sql*)

Execute sql-statement direclty.

Executes the given sql-statement without preparing or binding any parameter

Warning

* Cannot use if prepare statement has already defined. Use execute instead.
* Not recommended when the case of dynamic sql and the sql-statement is not for procedure but for direct access to table. Use prepare and execute instead.

Parameters

sql	sql-statement you want to execute

Exceptions

exception	Error message from DBMS
exception	Method prepare is already called

6.130.3.4 auto fetch ( ) const -> bool

Fetch a record.

Returns

Whether succeded to fetch a record.

False means there's not any record or previous record was the last.

6.130.3.5 auto next ( ) const -> bool

Move cursor to the next sql-statement.

Returns

Whether succeded to move cursor to the next statement.

6.130.3.6 auto size ( ) const -> size_t

Get size of columns.

Returns the number of columns in a result set.

Warning

Not size of rows.

6.130.3.7 auto at (size_t *index*) const -> _Ty [inline]

Get column's data by its index.

Returns column's data from fetched-record by specified column index

Parameters

index	Index number of a column wants to get

Returns

Data stored in the record at the position of specifield column

6.130.3.8 auto get (const std::string &) const -> _Ty [inline]

Get a column data by its name.

Returns column's data from fetchched-recrod by specified column name

Get data from fetched-record by specified column name

Parameters

name	Name of a column wants to get

Returns

Data stored in the record at the position of specifield column

6.130.3.9 auto toXML ( ) const -> std::shared_ptr<XML> [virtual]

Result sets to XML.

Converts the records of current sql-statement to XML.

Recommends to override for each DBMS's domain XML rule.

Returns

XML representing records of the statement

Reimplemented in TSQLStatement.

6.130.4 Member Data Documentation

6.130.4.1 Map<size_t, SQL_SIZE_T> bindParameterBASizeMap [protected]

A map for binary size.

When calls ByteArray::size(), the returned size_t value can't be kept until

The documentation for this class was generated from the following files:

* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SQLStatement.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/SQLStatement.cpp

6.131 StringUtil Class Reference

Utility class for string.

`#include <StringUtil.hpp>`

Static Public Member Functions

* template<typename _Ty , typename... _Args>
  static auto substitute (const std::string &format, const _Ty &val, const _Args &...args) -> std::string
  Substitutes "{n}" tokens within the specified string with the respective arguments passed in.
* template<typename _Ty , typename... _Args>
  static auto substituteSQL (const std::string &format, const _Ty &value, const _Args &...args) -> std::string
  Substitutes "{n}" tokens within the specified sql-string with the respective arguments passed in.
  .
* static auto isNumeric (const std::string &str) -> bool
  Returns wheter the std::string represents Number or not
  .
* static auto toNumber (const std::string &str) -> double
  Number std::string to Number having ',' symbols.
* static auto numberFormat (double val, int precision=2) -> std::string
* static auto percentFormat (double val, int precision=2) -> std::string
  Returns a percentage string converted from the number rounded off from specified precision with "," symbols
  ex) percentFormat(11.3391, 1) => 1,133.9%.
* static auto colorNumberFormat (double value, int precision=2, double delimiter=0.0) -> std::string
  Returns a string converted from the number rounded off from specified precision with "," symbols and color tag
  ex) numberFormat(17151.339, 2) => <font color="red">17,151.34</font>
* static auto colorPercentFormat (double value, int precision=2, double delimiter=0.0) -> std::string
  Returns a percentage string converted from the number rounded off from specified precision with "," symbols
  ex) percentFormat(11.3391, 1) => 1,133.9%.
* static auto trim (const std::string &val, const std::vector< std::string > &delims) -> std::string
  Removes all designated characters from the beginning and end of the specified string.
* static auto ltrim (const std::string &val, const std::vector< std::string > &delims) -> std::string
  Removes all designated characters from the beginning of the specified string.
* static auto rtrim (const std::string &val, const std::vector< std::string > &delims) -> std::string
  Removes all designated characters from the end of the specified string.
* static auto finds (const std::string &str, const std::vector< std::string > &delims, size_t start↩Index=0) -> IndexPair< std::string >

Finds first occurence in string.

* static auto rfinds (const std::string &str, const std::vector< std::string > &delims, size_t end↩ Index=SIZE_MAX) -> IndexPair< std::string >

  Finds last occurence in string.
* static auto substring (const std::string &str, size_t startIndex, size_t endIndex=SIZE_MAX) -> std↩ ::string

  Generates a substring.
* static auto between (const std::string &str, const std::string &start="", const std::string &end="") -> std::string

  Generates a substring.
* static auto addTab (const std::string &str, size_t n=1) -> std::string

  Adds tab() character to first position of each line.
* static auto split (const std::string &str, const std::string &delim) -> std::vector< std::string >

  Generates substrings.
* static auto betweens (const std::string &str, const std::string &start="", const std::string &end="") -> std::vector< std::string >

  Generates substrings.
* static auto toLowerCase (const std::string &str) -> std::string

  Returns a string that all uppercase characters are converted to lowercase

  .
* static auto toUpperCase (const std::string &str) -> std::string
* static auto replaceAll (const std::string &str, const std::string &before, const std::string &after) -> std::string

  Returns a string specified word is replaced.
* static auto replaceAll (const std::string &str, const std::vector< std::pair< std::string, std::string >> &pairs) -> std::string

  Returns a string specified words are replaced.

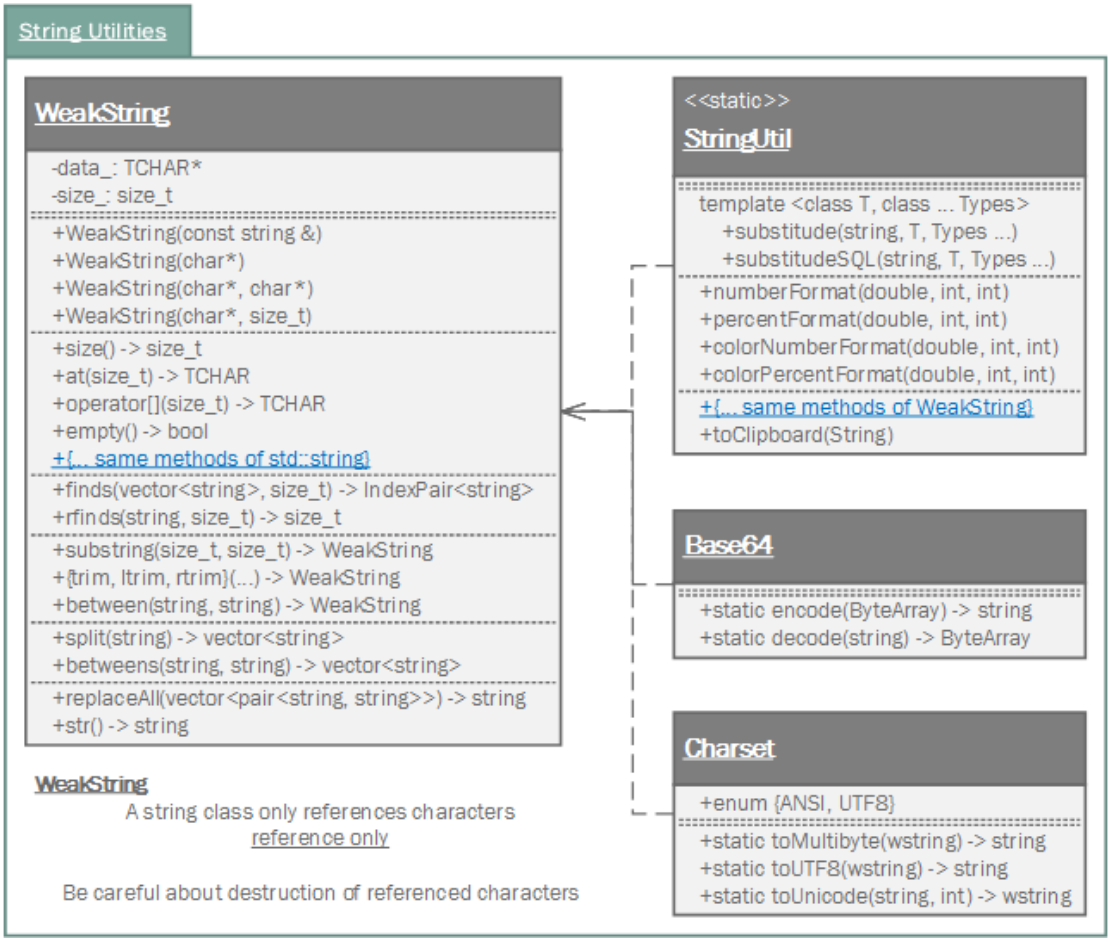
6.131.1 Detailed Description

Utility class for string.

StringUtil is an utility class providing lots of static methods for std::string.

There are two methods to strength std::string to have addictional uility methods like trim and split. The way of first is to make std::string class inheriting from std::string. The second is to make StringUtil class having static methods.

But those methods have problems. std::string class violates standard and StringUtil class violates principle of Object-Oriented Design. For the present, I've made the StringUtil class, but if you have a good opinion about the issue, please write your opinion on my github.



See also

library::WeakString
samchon::library

Author

Jeongho Nam

6.131.2 Member Function Documentation

6.131.2.1 static auto substitute (const std::string & *format,* const _Ty & *val,* const _Args &... *args*) -> std::string [inline], [static]

Substitutes "{n}" tokens within the specified string with the respective arguments passed in.

Parameters

format	The string to make substitutions in. This string can contain special tokens of the form {n}, where n is a zero based index, that will be replaced with the additional parameters found at that index if specified
val	Target value to substitute the minimum {n} tokens
args	Additional parameters that can be substituted in the str parameter at each {n} location, where n is an integer (zero based) index value into the varadics of values specified.

Returns

New string with all of the {n} tokens replaced with the respective arguments specified.

6.131.2.2 static auto substituteSQL (const std::string & *format,* const _Ty & *value,* const _Args &... *args*) -> std::string [inline], [static]

Substitutes "{n}" tokens within the specified sql-string with the respective arguments passed in.
.

Warning

substituteSQL creates the dynamic sql-statement.
Not recommended when the dynamic sql-statement is not only for procedure.

Parameters

format	The string to make substitutions in. This string can contain special tokens of the form {n}, where n is a zero based index, that will be replaced with the additional parameters found at that index if specified
val	Target value to substitute the minimum {n} tokens
args	Additional parameters that can be substituted in the str parameter at each {n} location, where n is an integer (zero based) index value into the varadics of values specified.

Returns

New sql-string with all of the {n} tokens replaced with the respective arguments specified.

6.131.2.3 auto isNumeric (const std::string & *str*) -> bool [static]

Returns wheter the std::string represents Number or not
.

Parameters

str	Target std::string to check

Returns

Whether the std::string can be converted to Number or not

6.131.2.4 auto toNumber (const std::string & *str*) -> double [static]

Number std::string to Number having ',' symbols.

Parameters

str	Target std::string you want to convert to Number

Returns

Number from std::string

6.131.2.5 auto numberFormat (double *val,* int *precision* = 2) -> std::string [static]

Returns a string converted from the number rounded off from specified precision with "," symbols ex) numberFormat(17151.339, 2) => 17,151.34

Parameters

val	A number wants to convert to string

precision	Target precision of roundoff

Returns

A string representing the number with roundoff and "," symbols

6.131.2.6 auto percentFormat (double *val,* int *precision* = 2) -> std::string [static]

Returns a percentage string converted from the number rounded off from specified precision with "," symbols
ex) percentFormat(11.3391, 1) => 1,133.9%.

Warning

Do not multiply by 100 to the value representing percent

Parameters

val	A number wants to convert to percentage string
precision	Target precision of roundoff

6.131.2.7 auto colorNumberFormat (double *value,* int *precision* = 2, double *delimiter* = 0.0) -> std::string [static]

Returns a string converted from the number rounded off from specified precision with "," symbols and color tag
ex) numberFormat(17151.339, 2) => <font color="red">17,151.34</font>
Which color would be chosen

* Number is positive, color is RED
* Number is zero (0), color is BLACK
* Number is negative, color is BLUE

Parameters

val	A number wants to convert to colored string
precision	Target precision of roundoff

Returns

A colored string representing the number with roundoff and "," symbols

6.131.2.8 auto colorPercentFormat (double *value,* int *precision* = 2, double *delimiter* = 0.0) -> std::string [static]

Returns a percentage string converted from the number rounded off from specified precision with "," symbols
ex) percentFormat(11.3391, 1) => 1,133.9%.

Warning

Do not multiply by 100 to the value representing percent

Parameters

val	A number wants to convert to percentage string
precision	Target precision of roundoff

6.131.2.9 static auto trim (const std::string & *val,* const std::vector< std::string > & *delims*) -> std::string [static]

Removes all designated characters from the beginning and end of the specified string.

Parameters

str	The string should be trimmed
delims	Designated character(s)

Returns

Updated string where designated characters was removed from the beginning and end

6.131.2.10 static auto ltrim (const std::string & *val,* const std::vector< std::string > & *delims*) -> std::string [static]

Removes all designated characters from the beginning of the specified string.

Parameters

str	The string should be trimmed
delims	Designated character(s)

Returns

Updated string where designated characters was removed from the beginning

6.131.2.11 static auto rtrim (const std::string & *val,* const std::vector< std::string > & *delims*) -> std::string [static]

Removes all designated characters from the end of the specified string.

Parameters

str	The string should be trimmed
delims	Designated character(s)

Returns

Updated string where designated characters was removed from the end

6.131.2.12 auto finds (const std::string & *str,* const std::vector< std::string > & *delims,* size_t *startIndex* = 0) -> IndexPair<std::string> [static]

Finds first occurence in string.

Finds first occurence position of each delim in the string after startIndex and returns the minimum position of them

If startIndex is not specified, then starts from 0.
If failed to find any substring, returns -1 (std::string::npos)

Parameters

str	Target string to find
delims	The substrings of target(str) which to find
startIndex	Specified starting index of find. Default is 0

Returns

pair<size_t := position, string := matched substring>

6.131.2.13 auto rfinds (const std::string & *str,* const std::vector< std::string > & *delims,* size_t *endIndex* = SIZE_MAX) -> IndexPair<std::string> [static]

Finds last occurence in string.

Finds last occurence position of each delim in the string before endIndex and returns the maximum position of them

If index is not specified, then starts str.size() - 1
If failed to find any substring, returns -1 (std::string::npos)

Parameters

str	Target string to find
delims	The substrings of target(str) which to find
endIndex	Specified starting index of find. Default is str.size() - 1

Returns

pair<size_t := position, string := matched substring>

6.131.2.14 auto substring (const std::string & *str*, size_t *startIndex*, size_t *endIndex* = `SIZE_MAX`) -> std::string `[static]`

Generates a substring.

Extracts a string consisting of the character specified by startIndex and all characters up to endIndex - 1 If endIndex is not specified, string::size() will be used instead.
If endIndex is greater than startIndex, then those will be swapped

Parameters

str	Target string to be applied substring
startIndex	Index of the first character. If startIndex is greater than endIndex, those will be swapped
endIndex	Index of the last character - 1. If not specified, then string::size() will be used instead

Returns

Extracted string by specified index(es)

6.131.2.15 auto between (const std::string & *str*, const std::string & *start* = " ", const std::string & *end* = " ") -> std::string `[static]`

Generates a substring.

Extracts a substring consisting of the characters from specified start to end It's same with str.substring(? = (str.find(start) + start.size()), str.find(end, ?))
ex) between("ABCD[EFGH]IJK", "[", "]") => "EFGH"

If start is not specified, extracts from begin of the string to end
If end is not specified, extracts from start to end of the string
If start and end are all omitted, returns str, itself.

Parameters

str	Target string to be applied between
start	A string for separating substring at the front
end	A string for separating substring at the end

Returns

substring by specified terms

6.131.2.16 auto addTab (const std::string & *str*, size_t *n* = `1`) -> std::string `[static]`

Adds tab() character to first position of each line.

Parameters

str	Target str to add tabs

n	The size of tab to be added for each line

Returns

A string added multiple tabs

6.131.2.17 auto split (const std::string & *str,* const std::string & *delim*) -> std::vector<std::string> [static]

Generates substrings.
Splits a string into an array of substrings by dividing the specified delimiter

Parameters

str	Target string to split
delim	The pattern that specifies where to split this string

Returns

An array of substrings

6.131.2.18 auto betweens (const std::string & *str,* const std::string & *start* = " ", const std::string & *end* = " ") -> std::vector<std::string> [static]

Generates substrings.
Splits a string into an array of substrings dividing by specified delimeters of start and end
It's the array of substrings adjusted the between

If startStr is omitted, it's same with the split by endStr not having last item
If endStr is omitted, it's same with the split by startStr not having first item
If startStr and endStar are all omitted, returns {str}

Parameters

str	Target string to split by between
start	A string for separating substring at the front If omitted, it's same with split(end) not having last item
end	A string for separating substring at the end If omitted, it's same with split(start) not having first item

Returns

An array of substrings

6.131.2.19 auto toLowerCase (const std::string & *str*) -> std::string [static]

Returns a string that all uppercase characters are converted to lowercase
.

Parameters

str	Target string to convert uppercase to lowercase

Returns

A string converted to lowercase

6.131.2.20 auto toUpperCase (const std::string & *str*) -> std::string [static]

Returns a string all lowercase characters are converted to uppercase

Parameters

str	Target string to convert lowercase to uppercase

Returns

A string converted to uppercase

6.131.2.21 static auto replaceAll (const std::string & *str,* const std::string & *before,* const std::string & *after*) -> std::string [static]

Returns a string specified word is replaced.

Parameters

str	Target string to replace
before	Specific word you want to be replaced
after	Specific word you want to replace

Returns

A string specified word is replaced

6.131.2.22 static auto replaceAll (const std::string & *str,* const std::vector< std::pair< std::string, std::string >> & *pairs*) -> std::string [static]

Returns a string specified words are replaced.

Parameters

str	Target string to replace
pairs	A specific word's pairs you want to replace and to be replaced

Returns

A string specified words are replaced

The documentation for this class was generated from the following files:

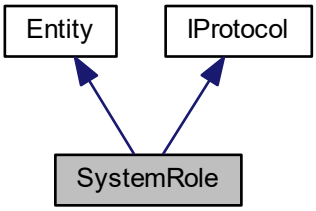
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/StringUtil.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/StringUtil.cpp

6.132 SystemRole Class Reference

A role belongs to a system.

`#include <SystemRole.hpp>`

Collaboration diagram for SystemRole:



Public Member Functions

* SystemRole ()
  Default Constructor.
* virtual void construct (std::shared_ptr< library::XML >) override
  Construct data of the Entity from an XML object.
* virtual auto key () const -> std::string override
  Get a key that can identify the Entity uniquely.
* auto hasListener (const std::string &) const -> bool
  Test whether has a listener in the role.
* virtual auto TAG () const -> std::string override
  A tag name when represented by XML.
* virtual auto toXML () const -> std::shared_ptr< library::XML > override
  Get an XML object represents the Entity.

Protected Attributes

* std::string name
  A name representing the role.
* std::set< std::string > listeners
  Listeners belongs to the role.

6.132.1 Detailed Description

A role belongs to a system.

SystemRole is an abstract and a 'control' class groupping methods, handling Invoke messages interacting with an system that the SystemRole is belonged to, by a subject or unit of a module.

Author

Jeongho Nam

6.132.2 Member Function Documentation

6.132.2.1 void construct (std::shared_ptr< library::XML >) [override], [virtual]

Construct data of the Entity from an XML object.

Overrides the construct() method and fetch data of member variables from the XML.

By recommended guidance, data representing member variables are contained in properties of the put XML object.

Parameters

xml	An xml used to construct data of entity

Implements Entity.

6.132.2.2 auto key ( ) const -> std::string [override], [virtual]

Get a key that can identify the Entity uniquely.

If identifier of the Entity is not atomic value, returns a string represents the composite identifier. If identifier of the Entity is not string, converts the identifier to string and returns the string.

Returns

An identifier

Reimplemented from Entity.

6.132.2.3 auto TAG ( ) const -> std::string [override], [virtual]

A tag name when represented by XML.

Returns

A tag name

Implements Entity.

6.132.2.4 auto toXML ( ) const -> std::shared_ptr<library::XML> [override], [virtual]

Get an XML object represents the Entity.

Returns an XML object that can represents the Entity containing member variables into properties.

A member variable (not object, but atomic value like number, string or date) is categorized as a property within the framework of entity side. Thus, when overriding a toXML() method and archiving member variables to an XML object to return, puts each variable to be a property belongs to only an XML object.

Don't archive the member variable of atomic value to XML::value causing enormouse creation of XML objects to number of member variables. An Entity must be represented by only an XML instance (tag).

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

Returns

An XML object representing the Entity.

Reimplemented from Entity.

The documentation for this class was generated from the following files:

* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/SystemRole.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/SystemRole.cpp

6.133 Travel Class Reference

Represent a travel containning Point(s)

`#include <Travel.hpp>`

Public Member Functions

* Travel ()
  Default Constructor.
* Travel (const Travel &)
  Copy Constructor.
* Travel (Travel &&)
  Move Constructor.
* auto operator< (const Travel &) const -> bool
  Compare which object is less.
* auto toString () const -> std::string
  Convert the Travel to String.

Private Member Functions

* auto calcDistance () const -> double
  Calculate distance to move.

Private Attributes

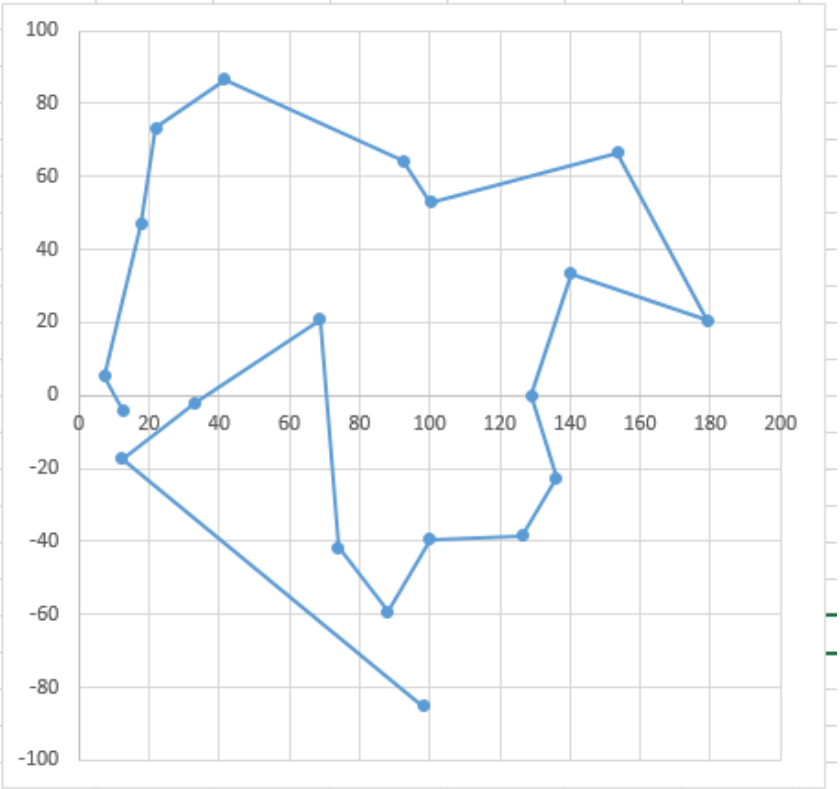
* double distance
  Estimated hours to move.

6.133.1 Detailed Description

Represent a travel containning Point(s)

Travel class is not only an array containing Point(s) but also a sequence listing of the Point(s).

Distance: 64,896.70 km		
uid	longitude	latitude
18	12.5233	-3.95529
15	7.23275	5.18342
7	17.7528	46.8665
2	22.0434	73.4133
20	41.6615	86.5372
13	92.427	64.3369
1	100.345	53.1067
9	153.521	66.5119
3	179.118	20.5073
10	140.113	33.3537
17	128.885	0.054607
4	136.012	-22.4915
6	126.352	-38.5031
16	99.9775	-39.3696
11	87.8859	-59.3663
14	73.8477	-41.5867
5	68.6563	20.8528
12	33.2162	-1.92823
8	12.1839	-17.3552
19	98.0967	-85.195



Author

Jeongho Nam

6.133.2 Constructor & Destructor Documentation

6.133.2.1 Travel (const Travel & *travel*)

Copy Constructor.

Travel is an array of Point(s) as gene(s). The copy constructor of Travel doesn't copy member variables which are related to G.A. because the sequence of gene(Point)s can be shuffled by process of evolution of genetic algorithm.

6.133.3 Member Function Documentation

6.133.3.1 auto operator< (const Travel & *travel*) const -> bool

Compare which object is less.

Compares distances to move. The result is used to optimize and evolve genes by G.A.

Parameters

Target	object to compare

Returns

Whether this object is less than target

6.133.3.2 auto toString ( ) const -> std::string

Convert the Travel to String.

Have of form of tab and enter delimeters for Excel.

```
{$uid1}     {$x}     {$y}
```

```
{$uid2}    {$x}    {$y}



{$uid3}    {$x}    {$y}
```

...

Returns

A string can represent the Travel

6.133.4 Member Data Documentation

6.133.4.1 double distance [private]

Estimated hours to move.

A variable for avoiding duplicated calculation

See also

calcDistance()

The documentation for this class was generated from the following files:

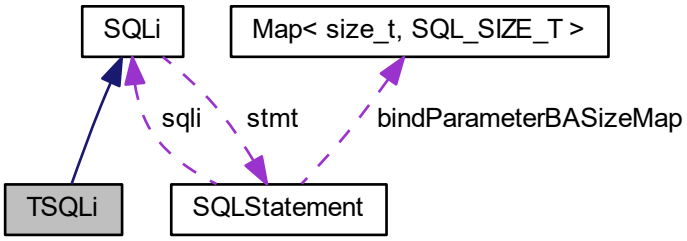
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/tsp/Travel.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/tsp/Travel.cpp

6.134 TSQLi Class Reference

A SQL interface for T-SQL.

`#include <TSQLi.hpp>`

Collaboration diagram for TSQLi:



Public Member Functions

* TSQLi (int port=1433)
  Construct from port.
* virtual auto createStatement () -> std::shared_ptr< SQLStatement >
  Factory method for creating T-SQL statement.

Additional Inherited Members

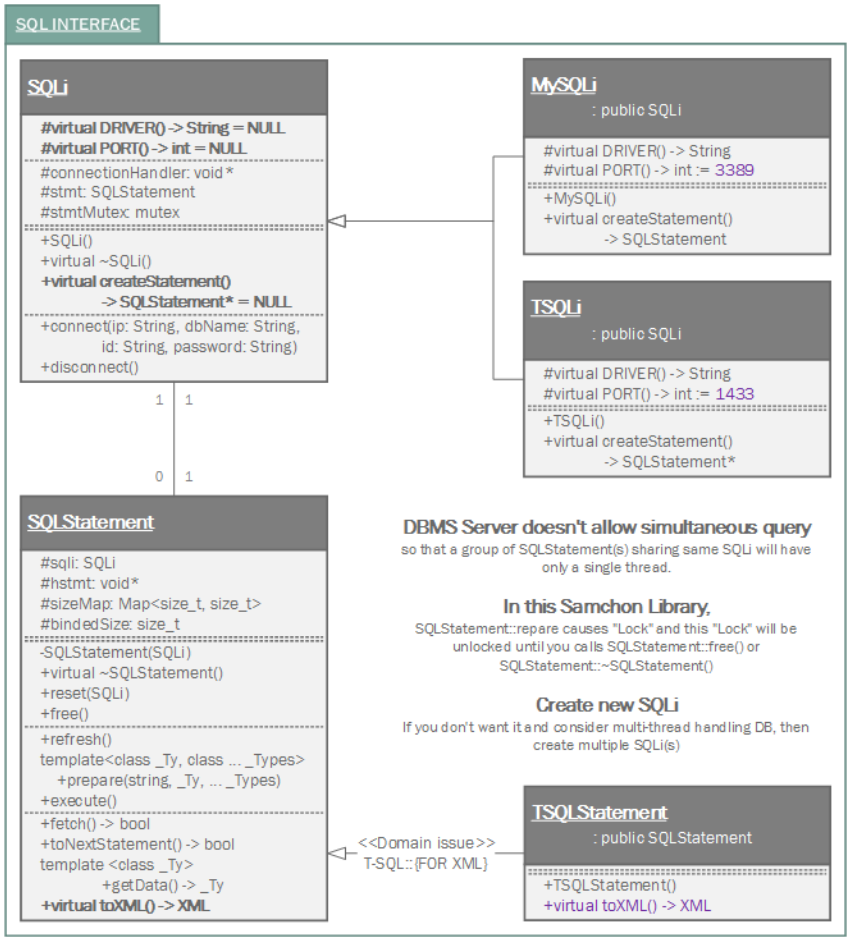
6.134.1 Detailed Description

A SQL interface for T-SQL.

[Inherited]

SQLi is an abstract class managing connection (session) with a specific database and manages SQL statements are executed and results are returned within the context of a connection.

The reason why SQLStatement(s) are managed by SQLi is because of a principle of DBMS, DBMS system doesn't allow simultaneous query from a session (connection). Only a query (process0) is allowed at a time. If you try to simultaneous query from a SQLi, ODBC throws error. To avoid the error, an execute of query from SQLStatement will lock a mutex of SQLi to ensure exclusiveness.



Note

To ensure the exclusiveness, you've to make SQLStatement from SQLi. Do not make SQLStatement by yourself. call SQLi::createSQLStatement() instead. Even you make a derived class from SQLStatement, don't make its constructor to have public accessor.

Becuase execution of a sql statement causes lock on mutex in SQLi, you've to destruct the SQLStatement or call SQLStatement::free() method(). If you don't, the mutex will not be unlocked, thus you can't do anything by the SQLi.

See also

samchon::library

Author

Jeongho Nam

6.134.2 Constructor & Destructor Documentation

6.134.2.1 TSQLi (int *port* = 1433)

Construct from port.

Parameters

port	Port number of T-SQL. default is 1433.

6.134.3 Member Function Documentation

6.134.3.1 auto createStatement ( ) -> std::shared_ptr<SQLStatement> [virtual]

Factory method for creating T-SQL statement.

To realize domain function in T-SQL, 'FOR XML' statement.

[Inherited]

SQLi::createStatement()

Returns

A T-SQL statement

Reimplemented from SQLi.

The documentation for this class was generated from the following files:

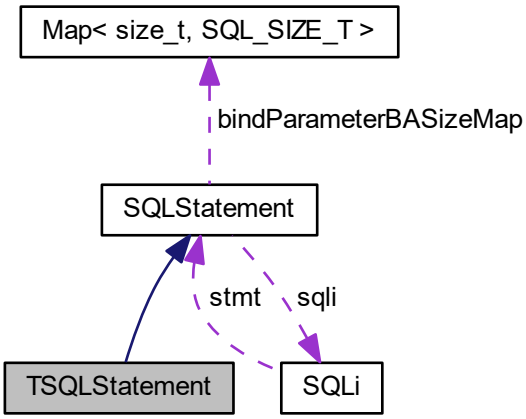
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/TSQLi.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/MSSQLi.cpp

6.135 TSQLStatement Class Reference

A t-sql statement.

`#include <TSQLStatement.hpp>`

Collaboration diagram for TSQLStatement:



Public Member Functions

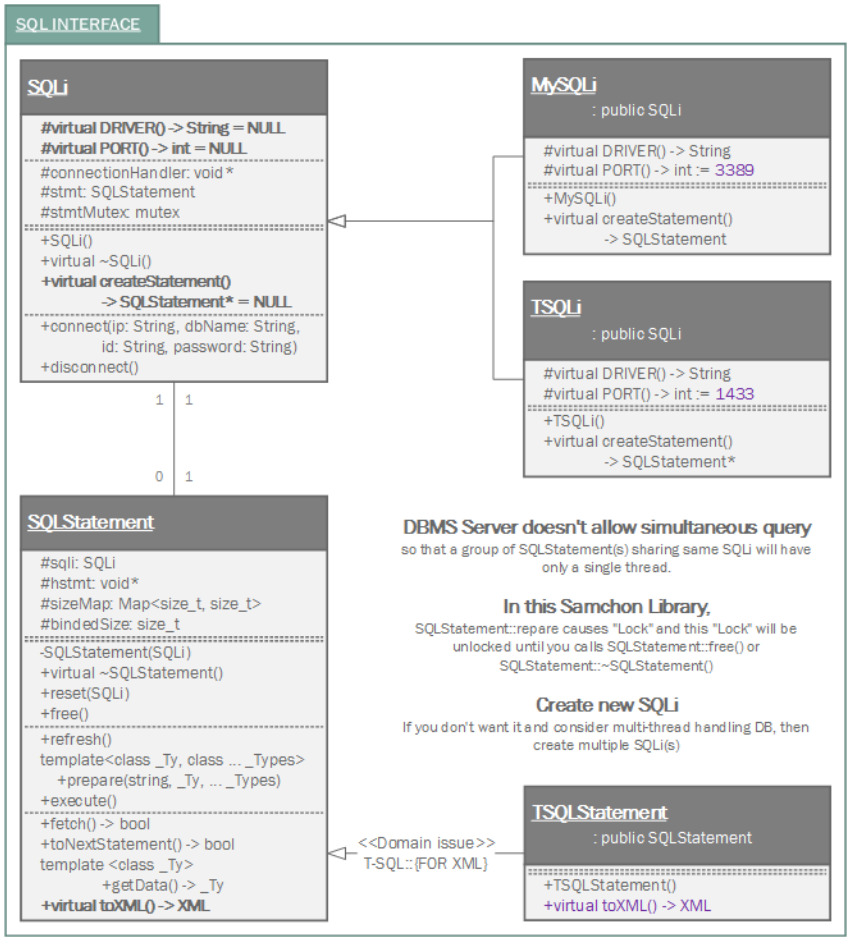
* virtual auto toXML () const -> std::shared_ptr< XML > override
  Return an XML representing records.

Additional Inherited Members

6.135.1 Detailed Description

A t-sql statement.
TSQLStatement is a SQLStatement for adjusting domain function f T-SQL, 'FOR XML' statement.
[Inherited]

A SQLStatement instance is used to executing a SQL statement and returning the results it produces against a SQL database that is opened through a SQLi instance.
Through the SQLi, it's the reason why a principle of DBMS, DBMS system doesn't allow simultaneous query from a session (connection). Only a query (process) is allowed at a time. If you try to simultaneous query from a SQLi, ODBC throws error. To avoid the error, an execute of query from SQLStatement will lock a mutex of SQLi to ensure exclusiveness.



Note

To ensure the exclusiveness, you've to make SQLStatement from SQLi. Do not make SQLStatement by yourself. call SQLi::createSQLStatement() instead. Even you make a derived class from SQLStatement, don't make its constructor to have public accessor.
Becuase execution of a sql statement causes lock on mutex in SQLi, you've to destruct the SQL↩ Statement or call SQLStatement::free() method(). If you don't, the mutex will not be unlocked, thus you can't do anything by the SQLi.

See also

samchon::library

Author

Jeongho Nam

6.135.2 Member Function Documentation

6.135.2.1 auto toXML ( ) const -> std::shared_ptr<XML> [override], [virtual]

Return an XML representing records.

Get an XML object generated from a sql statement containing 'FOR XML'

Returns

XML represents records

Reimplemented from SQLStatement.

The documentation for this class was generated from the following files:

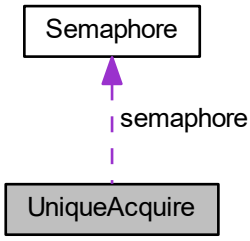
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/TSQLStatement.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/MSSQLStatement.cpp

6.136 UniqueAcquire Class Reference

Unique acquire from a Semaphore.

`#include <UniqueAcquire.hpp>`

Collaboration diagram for UniqueAcquire:



Public Member Functions

* UniqueAcquire (Semaphore &, bool=true)
  Construct from semaphore.
* UniqueAcquire (const UniqueAcquire &)=delete
  Prohibited Copy Constructor.
* UniqueAcquire (UniqueAcquire &&)
  Move Constructor.
* ~UniqueAcquire ()
  Default Destructor.
* void acquire ()
  Acquire admission.
* void release ()
  Release an admission.

Private Attributes

- Semaphore * semaphore
  Managed semaphore.
- bool isLocked
  Whether the semaphore was acquired by the UniqueLock.

6.136.1 Detailed Description

Unique acquire from a Semaphore.

A UniqueAcquire is an object manages a Semaphore with unique ownership in both states.

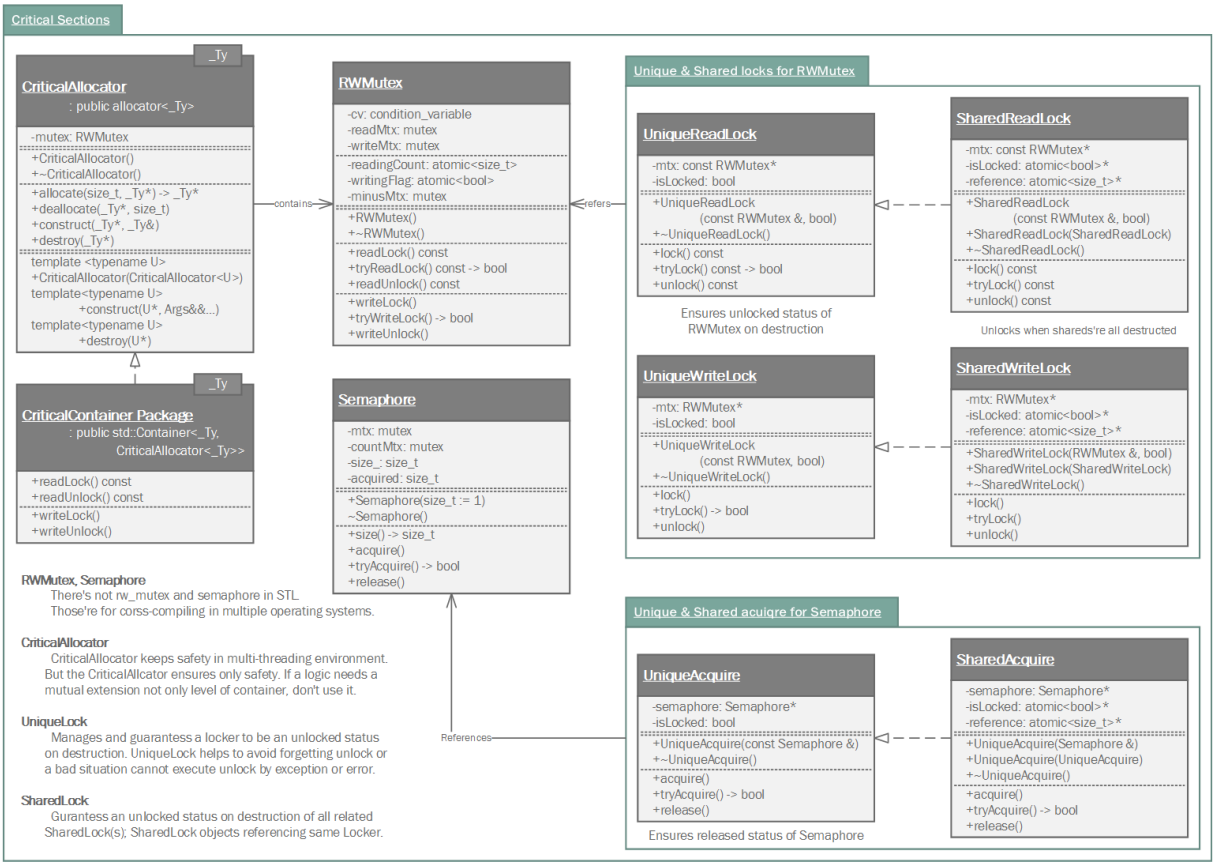
- acquired
- released

On construction (or by move-assigning to it), the object acquires a semaphore object, for whose acquiring and releasing operations becomes responsible.

This class guarantees a released status on destruction (even if not called explicitly). Therefore it is especially useful as an object with automatic duration, as it guarantees the semaphore object is properly released in case an exception is thrown.

Referenced comments of std::unique_lock

- http://www.cplusplus.com/reference/mutex/unique_lock/



Note

Though, that the UniqueAcquire object does not manage the lifetime of the semaphore object in any way: the duration of the semaphore object shall extend at least until the destruction of the Unique↩ Acquire that manages it.

The UniqueAcquire is movable, but not copyable.

See also

samchon::library

Author

Jeongho Nam

6.136.2 Constructor & Destructor Documentation

6.136.2.1 UniqueAcquire (Semaphore & *semaphore*, bool *doLock* = `true`)

Construct from semaphore.

Parameters

semaphore	Semaphore to manage
doLock	Whether to lock directly or not

6.136.2.2 UniqueAcquire (const UniqueAcquire &) `[delete]`

Prohibited Copy Constructor.

UniqueAcquire can't be copied. Use pointer, reference instead.

If what you want is UniqueAcquire(s) references each other and unlock when all related UniqueAcquire objects are destructed, SharedAcquire is the best way.

Parameters

obj	Tried object to copy.

6.136.2.3 UniqueAcquire (UniqueAcquire && *obj*)

Move Constructor.

Parameters

An	object to move

6.136.2.4 ~UniqueAcquire ( )

Default Destructor.

If read lock has done by the UniqueLock, unlock it

6.136.3 Member Function Documentation

6.136.3.1 void acquire ( )

Acquire admission.

Acquires an admission and increases count of admission by 1.

If the count is over permitted size, wait until other admissions to be released.

* Lock on mutex

6.136.3.2 void release ( )

Release an admission.

Releases an admission what you've acquired. If the admission count was over the limited size, unlock the mutex.

The documentation for this class was generated from the following files:

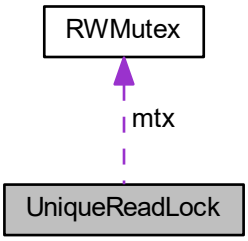
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/UniqueAcquire.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/UniqueAcquire.cpp

6.137 UniqueReadLock Class Reference

Unique lock for reading.

`#include <UniqueReadLock.hpp>`

Collaboration diagram for UniqueReadLock:



Public Member Functions

- UniqueReadLock (const RWMutex &, bool=true)
  Construct from mutex.
- UniqueReadLock (const UniqueReadLock &)=delete
  Prohibited Copy Constructor.
- UniqueReadLock (UniqueReadLock &&)
  Move Constructor.
- ~UniqueReadLock ()
  Default Destructor.
- void lock () const
  Lock on read.
- void unlock () const
  Unlock of read.

Private Attributes

- const RWMutex * mtx
  Managed mutex.
- bool isLocked
  Whether the mutex was locked by UniqueLock.

6.137.1 Detailed Description

Unique lock for reading.

A UniqueAcquire is an object manages a RWMutex with unique ownership in both states.

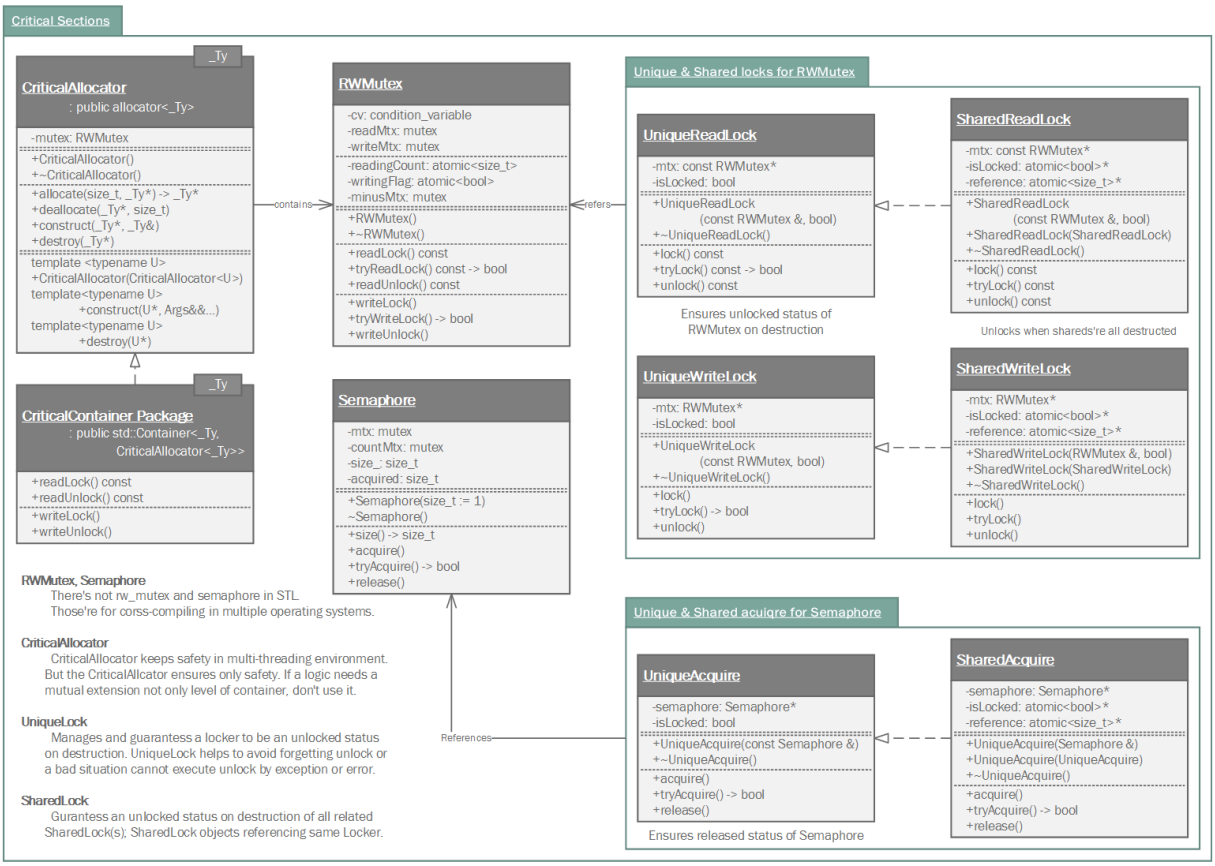
- locked
- unlocked

On construction (or by move-assigning to it), the object locks a RWMutex object on reading side, for whose locking and unlocking operations becomes responsible.

This class guarantees a unlocked status on destruction (even if not called explicitly). Therefore it is especially useful as an object with automatic duration, as it guarantees the RWMutex object is properly unlocked in case an exception is thrown.

Referenced comments of std::unique_lock

- http://www.cplusplus.com/reference/mutex/unique_lock/



Note

Though, that the UniqueReadLock object does not manage the lifetime of the RWMutex object in any way: the duration of the RWMutex object shall extend at least until the destruction of the UniqueReadLock that manages it.

See also

samchon::library

Author

Jeongho Nam

6.137.2 Constructor & Destructor Documentation

6.137.2.1 UniqueReadLock (const RWMutex & *mtx*, bool *doLock* = `true`)

Construct from mutex.

Parameters

mtx	Mutex to manage
doLock	Whether to lock directly or not

6.137.2.2 UniqueReadLock (const UniqueReadLock &) `[delete]`

Prohibited Copy Constructor.

UniqueAcquire can't be copied. Use pointer, reference instead.

If what you want is UniqueAcquire(s) references each other and unlock when all related UniqueAcquire objects are destructed, SharedAcquire is the best way.

Parameters

obj	Tried object to copy.

6.137.2.3 UniqueReadLock (UniqueReadLock && *obj*)

Move Constructor.

Parameters

An	object to move

6.137.2.4 ~UniqueReadLock ( )

Default Destructor.

If read lock has done by the UniqueLock, unlock it

6.137.3 Member Function Documentation

6.137.3.1 void lock ( ) const

Lock on read.

Increases a reading count.

When write_lock is on a progress, wait until write_unlock to be called.

* Reading can be done by multiple sections.
* Reading can't be done when writing.

Warning

You've to call read_unlock when the reading work is terminated.

6.137.3.2 void unlock ( ) const

Unlock of read.

Decreases a reading count.

When write_lock had done after read_lock, it continues by read_unlock if the reading count was 1 (read_unlock makes the count to be zero).

The documentation for this class was generated from the following files:

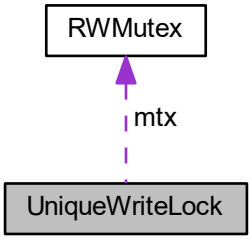
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/UniqueReadLock.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/UniqueReadLock.cpp

6.138 UniqueWriteLock Class Reference

Unique lock for writing.

```
#include <UniqueWriteLock.hpp>
```

Collaboration diagram for UniqueWriteLock:



Public Member Functions

- UniqueWriteLock (RWMutex &, bool=true)
  Construct from mutex.
- UniqueWriteLock (const UniqueWriteLock &)=delete
  Prohibited Copy Constructor.
- UniqueWriteLock (UniqueWriteLock &&)
  Move Constructor.
- ~UniqueWriteLock ()
  Destructor.
- void lock ()
  Lock on writing.
- void unlock ()
  Unlock on writing.

Private Attributes

- RWMutex * mtx
  Managed mutex.
- bool isLocked
  Whether the mutex was locked by UniqueLock.

6.138.1 Detailed Description

Unique lock for writing.

A UniqueWriteLock is an object manages a RWMutex with unique ownership in both states.

- acquired
- released

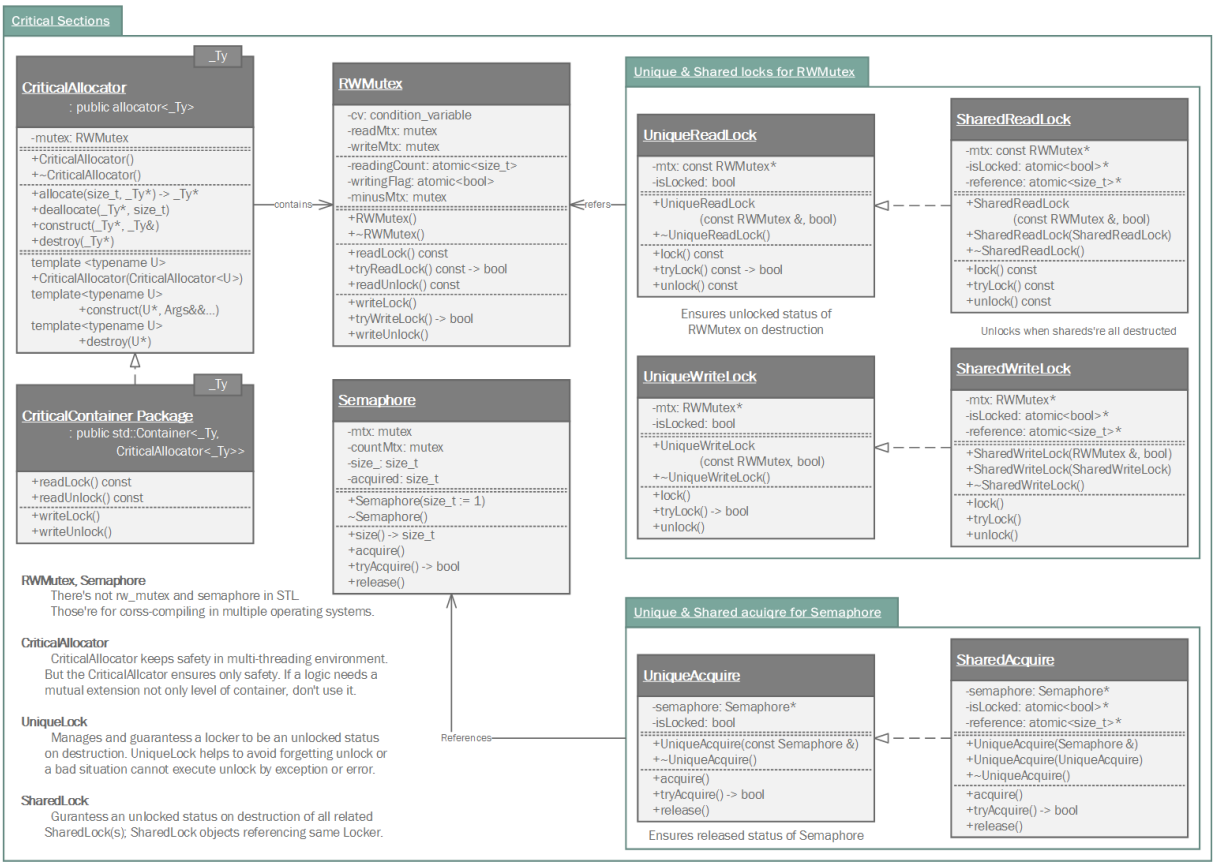
On construction (or by move-assigning to it), the object locks a RWMutex object on writing side, for whose locking and unlocking operations becomes responsible.

The object supports both states: locked and unlocked.

This class guarantees a unlocked status on destruction (even if not called explicitly). Therefore it is especially useful as an object with automatic duration, as it guarantees the RWMutex object is properly unlocked in case an exception is thrown.

Referenced comments of std::unique_lock

- http://www.cplusplus.com/reference/mutex/unique_lock/



Note

Though, that the UniqueWriteLock object does not manage the lifetime of the RWMutex object in any way: the duration of the RWMutex object shall extend at least until the destruction of the UniqueWriteLock that manages it.

See also

samchon::library

Author

Jeongho Nam

6.138.2 Constructor & Destructor Documentation

6.138.2.1 UniqueWriteLock (RWMutex & *mtx*, bool *doLock* = `true`)

Construct from mutex.

Parameters

mtx	Mutex to manage
doLock	Whether to lock directly or not

6.138.2.2 UniqueWriteLock (const UniqueWriteLock &) `[delete]`

Prohibited Copy Constructor.

UniqueAcquire can't be copied. Use pointer, reference instead.

If what you want is UniqueAcquire(s) references each other and unlock when all related UniqueAcquire objects are destructed, SharedAcquire is the best way.

Parameters

obj	Tried object to copy.

6.138.2.3 UniqueWriteLock (UniqueWriteLock && *obj*)

Move Constructor.

Parameters

An	object to move

6.138.2.4 ~UniqueWriteLock ( )

Destructor.

If write lock has done by the UniqueLock, unlock it

6.138.3 Member Function Documentation

6.138.3.1 void lock ( )

Lock on writing.

Changes writing flag to true.

If another write_lock or read_lock is on a progress, wait until them to be unlocked.

* Writing can be done by only a section at once.
* Writing can't be done when reading.

Note

You've to call write_unlock when writing work was terminated.

6.138.3.2 void unlock ( )

Unlock on writing.

The documentation for this class was generated from the following files:

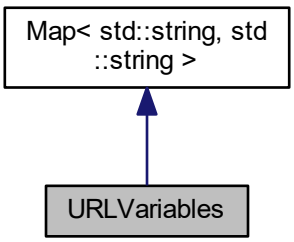
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/UniqueWriteLock.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/UniqueWriteLock.cpp

6.139 URLVariables Class Reference

URLVariables class is for representing variables of HTTP.

```
#include <URLVariables.hpp>
```

Collaboration diagram for URLVariables:



Public Member Functions

- URLVariables ()
  Default Constructor.
- URLVariables (const WeakString &flashVars)
  Constructor by a string representing encoded properties.
- auto toString () const -> std::string
  Get the string representing URLVariables.

Static Public Member Functions

- static auto encode (const WeakString &) -> std::string
  Encode a string into a valid URI.
- static auto decode (const WeakString &) -> std::string
  Decode a URI string.

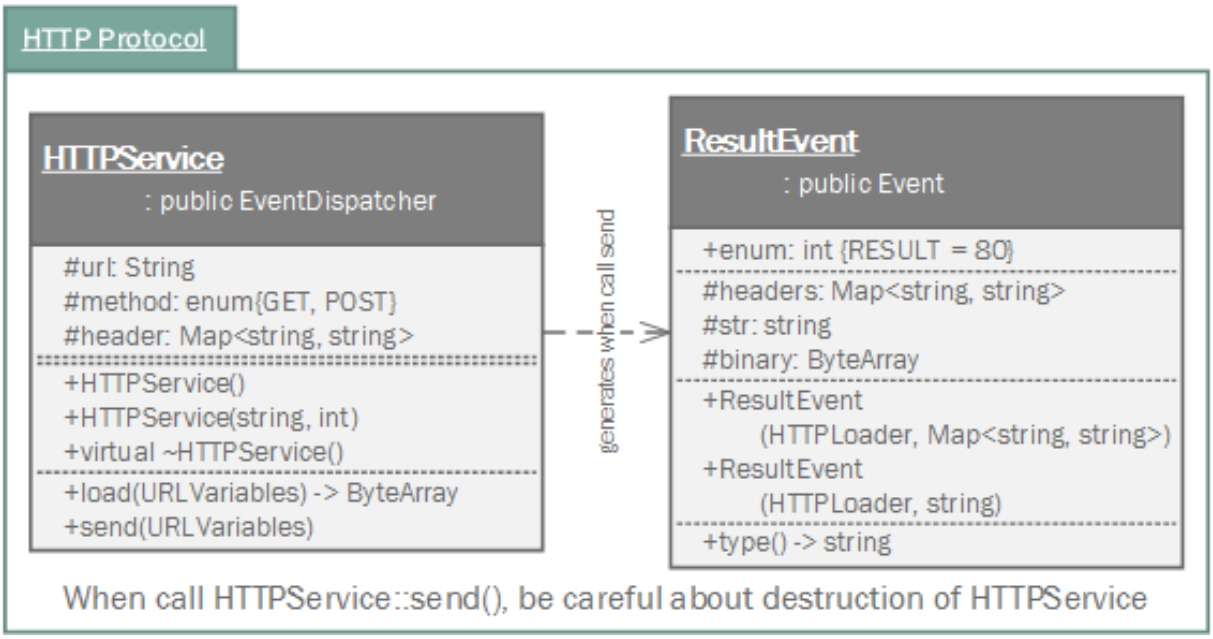
6.139.1 Detailed Description

URLVariables class is for representing variables of HTTP.

URLVariables class allows you to transfer variables between an application and server. When transferring, URLVariables will be converted to a URI string.

- URI: Uniform Resource Identifier

Use URLVariabels objects with methods of HTTPLoader class.



Example code

See also

samchon::library

Author

Jeongho Nam

6.139.2 Constructor & Destructor Documentation

6.139.2.1 URLVariables (const WeakString & *flashVars*)

Constructor by a string representing encoded properties.

Converts the variable string to properties of the specified URLVariables object.

* URLVariables("id=jhnam88&name=Jeongho+Nam") => {{\"id", "jhnam88"}, {"name", "Jeongho Nam"}}

Parameters

A	uri-encoded string containing pair of properties

6.139.3 Member Function Documentation

6.139.3.1 auto encode (const WeakString & *wstr*) -> std::string [static]

Encode a string into a valid URI.

Encodes a string to follow URI standard format.

Parameters

A	string to encode to URI

Returns

A string converted to URI

6.139.3.2 auto decode (const WeakString & *wstr*) -> std::string [static]

Decode a URI string.

Decodes a URI string to its original

Parameters

A	string encoded

Returns

A string decoded from URI

6.139.3.3 auto toString ( ) const -> std::string

Get the string representing URLVariables.

Returns a string object representing URLVariables following the URI

* URLVariables({{"id", "jhnam88"}, {"name", "Jeongho Nam"}}).toString() => "id=jhnam88&name=↩ Jeongho+Nam"

Returns

A string representing URLVariables following the URI

The documentation for this class was generated from the following files:

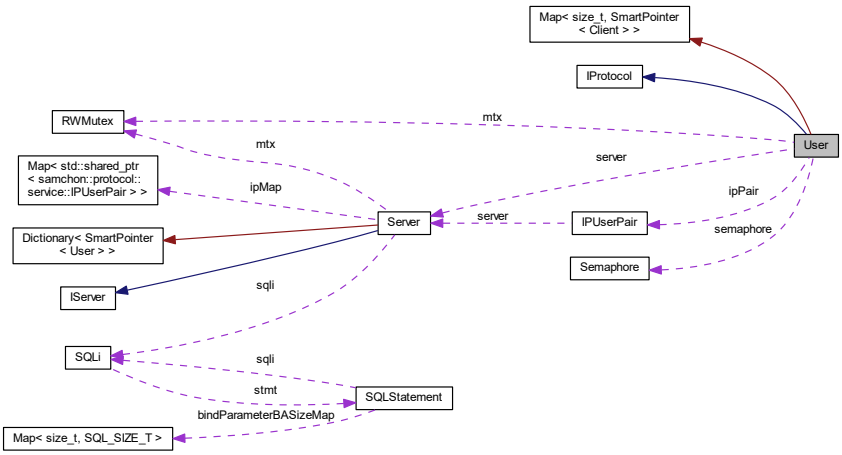
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/URLVariables.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/URLVariables.cpp

6.140 User Class Reference

User containing Client(s) with session-id.

```
#include <User.hpp>
```

Collaboration diagram for User:



Public Member Functions

- User (Server *)
  Construct from server.
- auto getServer () const -> Server *
  Get server.
- auto getSemaphore () const -> library::Semaphore *
  Get semaphore.
- auto getID () const -> std::string
  Get user's account id.
- auto getAuthority () const -> int
  Get authority of user.
- virtual void sendData (std::shared_ptr< Invoke >) override
  Send a message.
- virtual void replyData (std::shared_ptr< Invoke >) override
  Reply a message.

Protected Member Functions

- virtual auto createClient () -> Client *=0
  Factory method of a Client.
- virtual auto doLogin (std::shared_ptr< Invoke >) -> bool=0
  Login and return whether success.
- virtual auto doJoin (std::shared_ptr< Invoke >) -> bool=0
  Join and returns whether success.

Private Member Functions

- void addClient (Socket *)
  Add a client in user.
- void eraseClient (size_t)
  Erase a Client.
- void goLogin (std::shared_ptr< Invoke >)
  Log-in.
- void goJoin (std::shared_ptr< Invoke >)
  Join as a member.
- virtual void goLogout ()
  Log-out.

Private Attributes

- Server * server

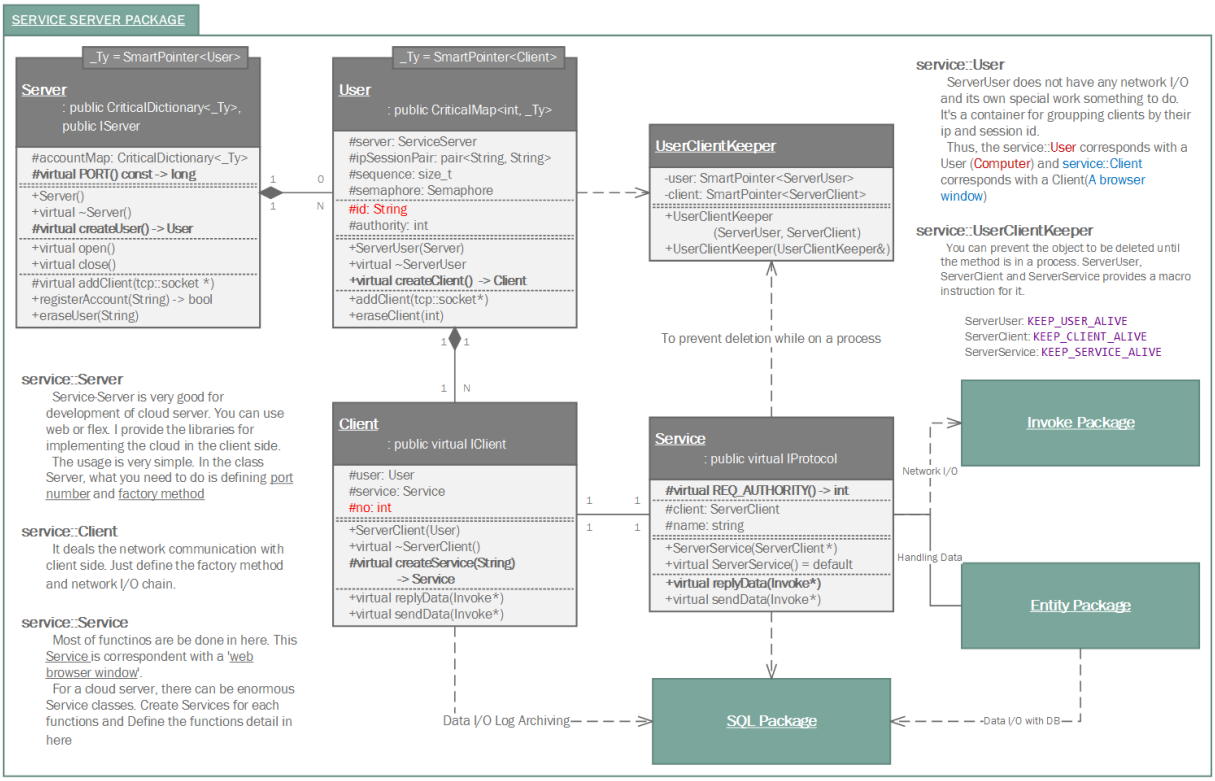
Server containing the user.

* std::string sessionID
  Session ID of the user.
* library::RWMutex mtx
  Mutex for container.
* library::Semaphore * semaphore
  Semaphore to limit number of thread.
* std::string id
  Account id.
* int authority
  Authority allocated to user.

6.140.1 Detailed Description

User containing Client(s) with session-id.

The User does not have any network I/O and its own special work something to do. It's a container for groupping clients by their session id. Thus, the User corresponds with an user (Computer) and Client corresponds with a client (A browser window)



Override those methods

* createClient()
* doLogin()
* doJoin()

Author

Jeongho Nam

6.140.2 Constructor & Destructor Documentation

6.140.2.1 User (Server * *server*)

Construct from server.

Parameters

server	A server who containing users and clients
sessionID	A session ID for discriminating User; A logical identifier

6.140.3 Member Function Documentation

6.140.3.1 auto getSemaphore ( ) const -> library::Semaphore*

Get semaphore.

See also

User::semaphore

6.140.3.2 auto getAuthority ( ) const -> int

Get authority of user.

See also

User::authority

6.140.3.3 void addClient (Socket * *socket*) [private]

Add a client in user.

Parameters

socket	Socket for interacting with (physical) client

6.140.3.4 void eraseClient (size_t *no*) [private]

Erase a Client.

Erase a Client from User by its key(no:int).

When a network connection between server and user(person) was disconnected, matched Client is erased from User and the Client object will be destructed.

Some reasons of disconnection

* An user(not object but person using UI) closed the Window
* A "closeWindow" command has sent to the Window
* A network failure has occured

If the Client was the last object remained in the User, the User will be destructed too. However, the User is not destructed directly but waits for 15 seconds. In the 15 seconds, another Client object was created in User, the User doesn't be destructed.

Some reasons of postponing destruction for 15 seconds

* Page direction(href) or redirection in a browser.
* Temporary network failure.

Note

There're some tasks that have to completed irrespectively of network connection.

In that case, although user closed the window or logged-out, matched User, Client and Service objects haven't be destructed. They must stay alive until the tasks have completed.

To stay those objects User, Client and Service, use follwing macros. Those macros assure the objects to stay alive until the termination matched each stack.

* KEEP_USER_ALIVE(Client*)
* KEEP_CLIENT_ALIVE
* KEEP_SERVICE_ALIVE

Parameters

no	Sequence number of the client in user

6.140.3.5 void goLogin (std::shared_ptr< Invoke >) [private]

Log-in.

Parameters

client	
invoke	

See also

User::doLogin()

6.140.3.6 void goJoin (std::shared_ptr< Invoke >) [private]

Join as a member.

Parameters

client	An application; who requested the join command
invoke	AN invoke message containing parameters of joinning

See also

User::doJoin()

6.140.3.7 void goLogout ( ) [private], [virtual]

Log-out.

Log-out account; change id and authority to "guest" and 0.

Default is to send "closeWindow" message to all clients except the applicant.

* Perspective on UI:
  All windows of the user will be closed by getting an invoke message "closeWindow" except who requested the command log-out.
  The applicant window will moved to default log-in page.

Parameters

client	A window of browser; client who requested the log-out command

6.140.3.8 virtual auto doLogin (std::shared_ptr< Invoke >) -> bool [protected], [pure virtual]

Login and return whether success.

Parameters

invoke	Invoke message containing parameters for logging-in

Returns

Whether the log-in is succeded or not

6.140.3.9 virtual auto doJoin (std::shared_ptr< Invoke >) -> bool [protected], [pure virtual]

Join and returns whether success.

Parameters

invoke	Invoke message containing paramters for joinning

Returns

Whether the join is succeded or not

6.140.3.10 void sendData (std::shared_ptr< Invoke >) [override], [virtual]

Send a message.

Sends Invoke message to a network system or shifts the responsibility to related chain.

Parameters

invoke	An Invoke message to send to a network system

Reimplemented from IProtocol.

6.140.3.11 void replyData (std::shared_ptr< Invoke >) [override], [virtual]

Reply a message.

Handles a replied Invoke message or shifts the responsibility to related chain.

Parameters

invoke	An Invoke message gotten from a network system

Reimplemented from IProtocol.

6.140.4 Member Data Documentation

6.140.4.1 std::string sessionID [private]

Session ID of the user.

Organization of the Session ID

* {$name}::{$port}::{$sequence}::{$linux_time}

1. $name: Name of the Server; Server::NAME()
2. $port: Port number of the Server; Server::PORT()
3. $sequence: Sequence number of creation of User in Server
4. $linux_time: Linux time of current time.

   If the object Server is re-constructed, $sequence can be duplicated. $linux_time is to avoid the duplication so that ensures exclusiveness on session id

6.140.4.2 library::Semaphore* semaphore [private]

Semaphore to limit number of thread.

Semaphore in User is designed to constraint number of thread.

It's recommended to acquire the semaphore when a task from Client or Service occurs a new thread and you want to avoid exhausted allocation of new threads.

Note

If a task influences to all clients in a user globally, use lock instead of acquire

6.140.4.3 std::string id [private]

Account id.

Default is "guest"

6.140.4.4 int authority [private]

Authority allocated to user.

The authority means authority(right, power) level allocated to the user.

The default value is zero, which means not logged-in user.

Note

Each number of level is not defined clearly. Follwing table is just an example.

Level	Definition
-1	Banned member The -1 level member is banished by manager or policy
0	Guest, Not logged in
1	Unconfired member The member have applied to the join, but the join was not confirmed by manager or policy yet.
2	Confirmed member The member have joined and have already confirmed by manager or policy
3	Manager A manager of the application(cloud service). The manager can * Access to the monitoring service. (MonitorService) * Access historical log of members whose level are below Manager. * Banish or modify level of members whose level are below Manager. </u>
4	Administrator or super Manager Administrator is a Manager of Managers. An administrator can do same thing with Managers, but adminstrator even possible to target Managers as below.

The documentation for this class was generated from the following files:

* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/User.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/service/User.cpp

6.141 WeakString Class Reference

A string class only references characeters, reference only.

`#include <WeakString.hpp>`

Public Member Functions

* WeakString ()
  Default Constructor does not reference any character.
* WeakString (const char *data, size_t size)
  Constructor by characters with specified size.
* WeakString (const char *begin, const char *end)
  Constructor by characters of begin and end.

* WeakString (const char *data)
  Constructor by characters.
* WeakString (const char &ch)
  Constructor by a single character.
* WeakString (std::initializer_list< char > &il)
  Constructor by a initializer list.
* WeakString (const std::string &str)
  Constructor by string.
* auto data () const -> const char *
  Get string data; referenced characeters.
* auto size () const -> size_t
  Returns size of the characters which are being referenced.
* auto empty () const -> bool
  Tests wheter string is emtpy.
* auto at (size_t index) const -> const char &
  Get character of string Returns a const reference to the character at the specified position.
* auto operator[] (size_t index) const -> const char &
  Get character of string Returns a const reference to the character at the specified position.
* auto find (const WeakString &delim, size_t startIndex=NULL) const -> size_t
  Finds first occurence in string.
* auto rfind (const WeakString &delim, size_t endIndex=SIZE_MAX) const -> size_t
  Finds last occurence in string.
* auto finds (const std::vector< std::string > &delims, size_t startIndex=0) const -> IndexPair< WeakString >
  Finds first occurence in string.
* auto rfinds (const std::vector< std::string > &delims, size_t endIndex=SIZE_MAX) const -> IndexPair< WeakString >
  Finds last occurence in string.
* auto substr (size_t startIndex, size_t endIndex=SIZE_MAX) const -> WeakString
  Generates a substring.
* auto substring (size_t startIndex, size_t size=SIZE_MAX) const -> WeakString
  Generates a substring.
* auto between (const WeakString &start={}, const WeakString &end={}) const -> WeakString
  Generates a substring.
* auto split (const WeakString &delim) const -> std::vector< WeakString >
  Generates substrings.
* auto betweens (const WeakString &start={}, const WeakString &end={}) const -> std::vector< WeakString >
  Generates substrings.
* auto trim (const std::vector< std::string > &delims) const -> WeakString
  Removes all designated characters from the beginning and end of the specified string.
* auto ltrim (const std::vector< std::string > &delims) const -> WeakString
  Removes all designated characters from the beginning of the specified string.
* auto rtrim (const std::vector< std::string > &delims) const -> WeakString
  Removes all designated characters from the end of the specified string.
* auto replace (const WeakString &before, const WeakString &after) const -> std::string
  Replace portion of string once.
* auto replaceAll (const WeakString &before, const WeakString &after) const -> std::string
  Returns a string specified word is replaced.
* auto replaceAll (const std::vector< std::pair< std::string, std::string >> &pairs) const -> std::string
  Returns a string specified words are replaced.
* auto toLowerCase () const -> std::string
  Convert uppercase letters to lowercase.
* auto toUpperCase () const -> std::string
  Convert uppercase letters to lowercase.
* auto str () const -> std::string
  Get the string content.

Static Public Attributes

* static const size_t npos = -1
  Maximum value for size_t.

Private Attributes

* const char * data_
  Referenced characters's pointer of begining position.

* size_t size_
  (Specified) size of referenced characters

Static Private Attributes

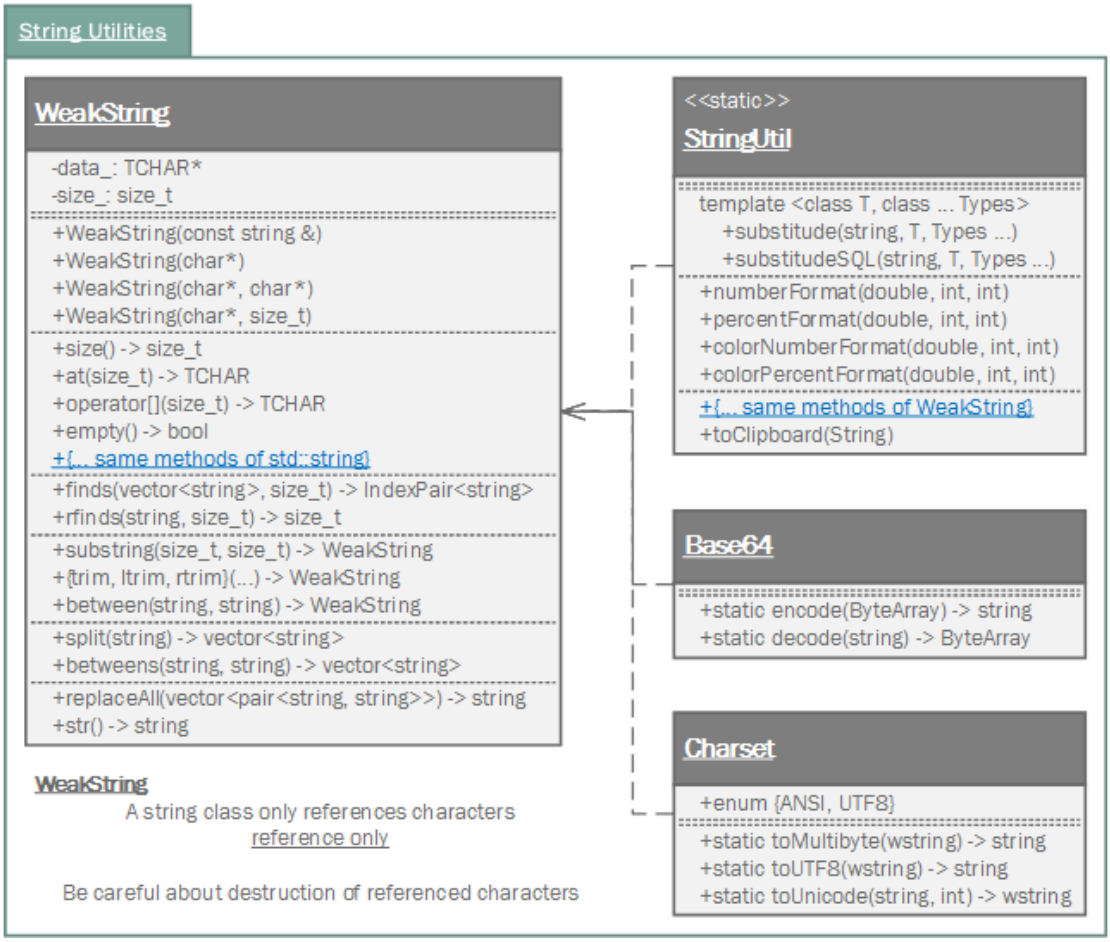
* static const std::vector< std::string > SPACE_ARRAY = { " ", "\t", "\r", "\n" }
  An array containing whitespaces.

6.141.1 Detailed Description

A string class only references characeters, reference only.

WeakSring does not consider any construction, modification and destruction of characters.

Thus, you can have greater advantages than std::string on the side of performance and memory, but of course, you can't modify the characeters at all.



Warning

* WeakString not copy(strcpy) characeters but only references characeters. Be careful about destruction of the characters being referenced by the WeakString
* WeakString will be used for basic data type in most case. Avoid to use WeakString by pre-definition in header (∗.hpp)

See also

samchon::library

Author

Jeongho Nam

6.141.2 Constructor & Destructor Documentation

6.141.2.1 WeakString ( )

Default Constructor does not reference any character.

Constructs an empty string, with zero size

6.141.2.2 WeakString (const char * *data*, size_t *size*)

Constructor by characters with specified size.

Constructs by characters to be referenced with limited size.

Although the original size of data is over the specified size, you can limit referencing size of the characters

* Referencing a part of characters

Warning

WeakString only references. Be careful about destruction of the characters (data)

Parameters

data	Target characters to be referenced by string
size	

Specified limit-size of characters to be referenced.

If the specified size is greater than original size, it will be ignored

6.141.2.3 WeakString (const char * *begin*, const char * *end*)

Constructor by characters of begin and end.

Constructs by characters to be referenced with its end position.

Although the original end point of data is over the specified end, you can limit end point of the characters.

* Referencing a part of characters

Warning

WeakString only references. Be careful about destruction of the characters (data).

Parameters

begin	Target characters to be referenced by string
end	

Specified end point of characters to be referenced.

If the specified end point is greater than original end point, it will be ignored.

6.141.2.4 WeakString (const char * *data*)

Constructor by characters.

References the null-terminated character sequence pointed by ptr

Warning

WeakString only references. Be careful about destruction of the characeters (data)

Parameters

data	Target characters to be referenced by string

6.141.2.5 WeakString (const char & *ch*)

Constructor by a single character.

References a single character

Warning

WeakString only references. Be careful about destruction of the characeter (data)

Parameters

ch	Target character to be referenced by string

6.141.2.6 WeakString (std::initializer_list< char > & *il*)

Constructor by a initializer list.

References initializer list of character

Warning

WeakString only references. Be careful about destruction of the characeters (data)

Parameters

il	Target initializer list of characters to be referenced by string

6.141.2.7 WeakString (const std::string & *str*)

Constructor by string.

References whole chracters of the string

Warning

WeakString only references. Be careful about destruction of the string

Parameters

str	Target string to be referenced by string

6.141.3 Member Function Documentation

6.141.3.1 auto data ( ) const -> const char*

Get string data; referenced characeters.

Returns a pointer to an array that contains a null-terminated sequence of characters representing the current value of the string object

Warning

Returned pointer's size can be longer than string's specified size if the string references only a part of the characters

Returns

A pointer of characters being referenced by the string

6.141.3.2 auto size ( ) const -> size_t

Returns size of the characters which are being referenced.

Returns the length of the string, in terms of number of referenced characters

Returns

size of characters being referenced by string

6.141.3.3 auto empty ( ) const -> bool

Tests wheter string is emtpy.
Returns wheter characters' size is zero or not.
Of course, string references nothing, then returns false, too.

Returns

Wheter size is zero or not

6.141.3.4 auto at (size_t *index*) const -> const char&

Get character of string Returns a const reference to the character at the specified position.

Returns

const reference of character at the specified index

6.141.3.5 auto operator[ ] (size_t *index*) const -> const char&

Get character of string Returns a const reference to the character at the specified position.

Returns

const reference of character at the specified index

6.141.3.6 auto find (const WeakString & *delim*, size_t *startIndex* = NULL) const -> size_t

Finds first occurence in string.
Finds the string after startIndex and returns the position of first occurence of delim.
If delim is not found, returns -1 (npos)

Parameters

delim	The substring of the string which to find
startIndex	Specified starting index of find. Default is 0

Returns

Index of first occurence of the specified substring or -1

6.141.3.7 auto rfind (const WeakString & *delim*, size_t *endIndex* = SIZE_MAX) const -> size_t

Finds last occurence in string.
Finds the string before endIndex and returns the position of last occurence of delim. If delim is not found, returns -1 (npos)

Parameters

delim	The substring of the string which to find
endIndex	Specified last index of find. Default is size() - 1

Returns

Index of first occurence of the specified substring or -1

6.141.3.8 auto finds (const std::vector< std::string > & *delims*, size_t *startIndex* = 0) const -> IndexPair<WeakString>

Finds first occurence in string.
Finds first occurence position of each delim in the string after startIndex and returns the minimum position of them.

Note

* If startIndex is not specified, then starts from 0.
* If failed to find any substring, returns -1 (npos)

Parameters

delims	The substrings of target(str) which to find
startIndex	Specified starting index of find. Default is 0

Returns

pair<size_t := position, string := matched substring>

6.141.3.9 auto rfinds (const std::vector< std::string > & *delims*, size_t *endIndex* = SIZE_MAX) const -> IndexPair<WeakString>

Finds last occurence in string.

Finds last occurence position of each delim in the string before endIndex and returns the maximum position of them.

Note

* If index is not specified, then starts str.size() - 1
* If failed to find any substring, returns -1 (npos)

Parameters

delims	The substrings of target(str) which to find
endIndex	Specified starting index of find. Default is size() - 1

Returns

pair<size_t := position, string := matched substring>

6.141.3.10 auto substr (size_t *startIndex*, size_t *endIndex* = SIZE_MAX) const -> WeakString

Generates a substring.

Extracts a substring consisting of the characters starts from startIndex and with a size specified size.

Parameters

startIndex	

Index of the first character.

If startIndex is greater than endIndex, those will be swapped.

Parameters

endIndex	

Number of characters to include in substring.

If the specified size is greater than last index of characeters, it will be shrinked.

Returns

Sub string by specified index and size

6.141.3.11 auto substring (size_t *startIndex*, size_t *size* = SIZE_MAX) const -> WeakString

Generates a substring.

Extracts a substring consisting of the character specified by startIndex and all characters up to endIndex - 1.

* If endIndex is not specified, string::size() will be used instead.
* If endIndex is greater than startIndex, then those will be swapped

Parameters

startIndex	

Index of the first character.

If startIndex is greater than endIndex, those will be swapped.

Parameters

size	

Index of the last character - 1.

If not specified, then string::size() will be used instead.

Returns

Sub string by specified index(es)

6.141.3.12 auto between (const WeakString & *start* = { }, const WeakString & *end* = { }) const -> WeakString

Generates a substring.

Extracts a substring consisting of the characters from specified start to end.

It's same with substring(? = (str.find(start) + start.size()), find(end, ?))

* between("ABCD[EFGH]IJK", "[", "]") => "EFGH"

Note

* If start is not specified, extracts from begin of the string to end
* If end is not specified, extracts from start to end of the string
* If start and end are all omitted, returns str, itself.

Parameters

start	A string for separating substring at the front
end	A string for separating substring at the end

Returns

substring by specified terms

6.141.3.13 auto split (const WeakString & *delim*) const -> std::vector<WeakString>

Generates substrings.

Splits a string in to an array of substrings dividing by the specified delimiter

Parameters

delim	The pattern which specifies where to split the string

Returns

An array of substrings

6.141.3.14 auto betweens (const WeakString & *start* = { }, const WeakString & *end* = { }) const -> std::vector<WeakString>

Generates substrings.

Splits a string into an array of substrings dividing by delimeters of start and end.

It's the array of substrings adjusted the between.

Note

* If start is omitted, it's same with the split by endStr not having last item
* If end is omitted, it's same with the split by startStr not having first item
* If start and end are all omitted, returns string, itself

Parameters

start	

A string for separating substring at the front.
If omitted, it's same with split(end) not having last item.

Parameters

end	

A string for separating substring at the end.
If omitted, it's same with split(start) not having first item.

Returns

An array of substrings

6.141.3.15 auto trim (const std::vector< std::string > & *delims*) const -> WeakString

Removes all designated characters from the beginning and end of the specified string.

Parameters

delims	Designated character(s)

Returns

Updated string where designated characters was removed from the beginning and end

6.141.3.16 auto ltrim (const std::vector< std::string > & *delims*) const -> WeakString

Removes all designated characters from the beginning of the specified string.

Parameters

delims	Designated character(s)

Returns

Updated string where designated characters was removed from the beginning

6.141.3.17 auto rtrim (const std::vector< std::string > & *delims*) const -> WeakString

Removes all designated characters from the end of the specified string.

Parameters

delims	Designated character(s)

Returns

Updated string where designated characters was removed from the end

6.141.3.18 auto replace (const WeakString & *before,* const WeakString & *after*) const -> std::string

Replace portion of string once.

Parameters

before	A specific word you want to be replaced
after	A specific word you want to replace

Returns

A string specific word is replaced once

6.141.3.19 auto replaceAll (const WeakString & *before,* const WeakString & *after*) const -> std::string

Returns a string specified word is replaced.

Parameters

before	A specific word you want to be replaced
after	A specific word you want to replace

Returns

A string specified word is replaced

6.141.3.20 auto replaceAll (const std::vector< std::pair< std::string, std::string >> & *pairs*) const -> std::string

Returns a string specified words are replaced.

Parameters

str	Target string to replace
pairs	A specific word's pairs you want to replace and to be replaced

Returns

A string specified words are replaced

6.141.3.21 auto toLowerCase ( ) const -> std::string

Convert uppercase letters to lowercase.

Returns a string that all uppercase characters are converted to lowercase.

Parameters

wstr	Target string to convert uppercase to lowercase

Returns

A string converted to lowercase

6.141.3.22 auto toUpperCase ( ) const -> std::string

Convert uppercase letters to lowercase.

Returns a string all lowercase characters are converted to uppercase.

Parameters

str	Target string to convert lowercase to uppercase

Returns

A string converted to uppercase

6.141.3.23 auto str ( ) const -> std::string

Get the string content.

Returns a string object with a copy of the current contents in the WeakString.

Returns

A new string copied from the WeakString

6.141.4 Member Data Documentation

6.141.4.1 const size_t npos = -1 [static]

Maximum value for size_t.

npos is a static member constant value with the greatest possible value for an element of type size_t.

This value, when used as the value for a len (or sublen) parameter in string's member functions, means "until the end of the string". As a return value, it is usually used to indicate no matches.

This constant is defined with a value of -1, which because size_t is an unsigned integral type, it is the largest possible representable value for this type.

The documentation for this class was generated from the following files:

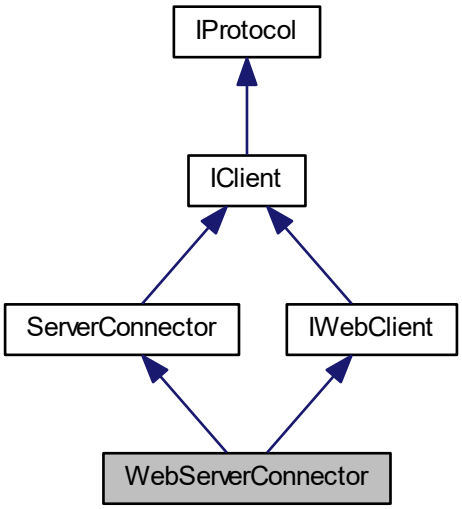
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/WeakString.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/WeakString.cpp

6.142 WebServerConnector Class Reference

A web-socket server connector.

`#include <WebServerConnector.hpp>`

Collaboration diagram for WebServerConnector:



Public Member Functions

* WebServerConnector ()
  Default Constructor.
* virtual void connect () override
  Connect to a server following web-socket protoocol.

Additional Inherited Members

6.142.1 Detailed Description

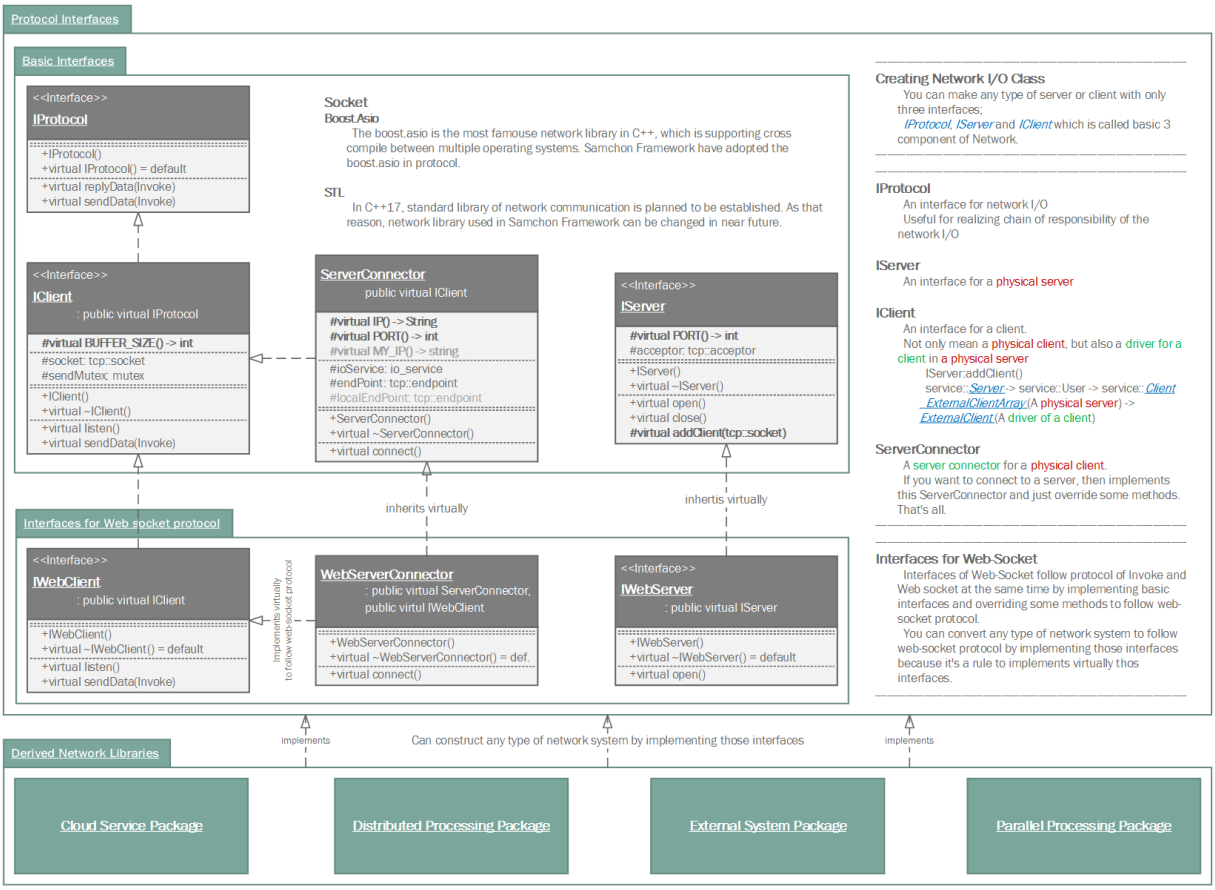
A web-socket server connector.

WebServerConnector is a ServerConnector following web-socket protocol.

[Inherited]

ServerConnector is a class for a physical client connecting a server. If you want to connect to a server, then implements this ServerConnector and just override some methods like getIP(), getPort() and replyData(). That's all.

In Samchon Framework, package protocol, There are basic 3 + 1 components that can make any type of network system in Samchon Framework. The basic 3 components are IProtocol, IServer and IClient. The last, surplus one is the ServerConnector. Looking around classes in Samchon Framework, especially module master and slave which are designed for realizing distributed processing systems and parallel processing systems, physical client classes are all derived from this ServerConnector.



Author

Jeongho Nam

The documentation for this class was generated from the following file:

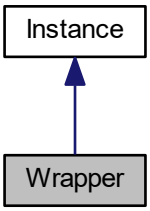
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/protocol/WebServerConnector.hpp

6.143 Wrapper Class Reference

A wrapper can contain products.

```
#include <Wrapper.hpp>
```

Collaboration diagram for Wrapper:



Public Member Functions

* Wrapper (const std::string &name, int price, int volume, int weight)
  Construct from argument of a wrapper.
* Wrapper (const Wrapper &wrapper)
  Copy Constructor.
* auto tryInsert (Product *) -> bool
  Try to insert a product into the wrapper.
* virtual auto toString () const -> std::string override
  Return a string represent the wrapper.

Additional Inherited Members

6.143.1 Detailed Description

A wrapper can contain products.

A wrapper containing products with its limitation of weight and volume.

```
D:\OneDrive\Project\Samchon\framework\cpp\x64\Release\packer.exe
$480
Category - Large
        Wrapper Large: $100, 200cm^3, 1000g
                Product Umbrella: $4000, 200cm^3, 1000g
        Wrapper Large: $100, 200cm^3, 1000g
                Product Eraser: $500, 10cm^3, 70g
                Product Pencil: $400, 30cm^3, 35g
                Product Notebook-PC: $800000, 150cm^3, 850g

Category - Medium
        Wrapper Medium: $70, 150cm^3, 500g
                Product Drink: $1000, 75cm^3, 250g
                Product Pencil: $400, 30cm^3, 35g
                Product Pencil: $400, 30cm^3, 35g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Book: $8000, 150cm^3, 300g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Book: $8000, 150cm^3, 300g
        Wrapper Medium: $70, 150cm^3, 500g
                Product Tablet-PC: $600000, 120cm^3, 450g

Category - Small


Press any key to continue . . .
```

Author

Jeongho Nam

6.143.2 Constructor & Destructor Documentation

6.143.2.1 Wrapper (const std::string & *name*, int *price*, int *volume*, int *weight*)

Construct from argument of a wrapper.

Parameters

name	Category name of a wrapper, can be identified
price	Price of a wrapper
volume	Limited volume of a wrapper can put in.
weight	Limited weight of a wrapper can put in.

6.143.2.2 Wrapper (const Wrapper & *wrapper*)

Copy Constructor.

Copy constructor of wrapper does not copy children items. Only copies arguments of Instance's.

6.143.3 Member Function Documentation

6.143.3.1 auto tryInsert (Product * *product*) -> bool

Try to insert a product into the wrapper.

If the Wrapper has enough volume and limit of weight, inserts the Product into the Wrapper and returns *true*. If not enough, does not insert and just return *false*.

Parameters

product	A product try to insert in.

Returns

Whether to success put in

6.143.3.2 auto toString ( ) const -> std::string [override], [virtual]

Return a string represent the wrapper.

Returns a string of the Wrapper and Product(s) packaged in.

Returns

A string represents the Wrapper and contained Product(s).

Reimplemented from Instance.

The documentation for this class was generated from the following files:

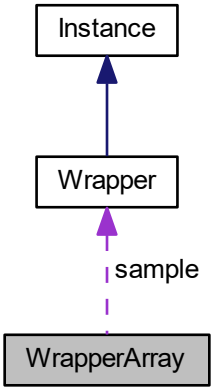
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/Wrapper.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/Wrapper.cpp

6.144 WrapperArray Class Reference

An array of wrapper with same category (name).

`#include <WrapperArray.hpp>`

Collaboration diagram for WrapperArray:



Public Member Functions

* WrapperArray (Wrapper *)
  Construct from a sample wrapper.
* auto tryInsert (Product *) -> bool
  Try to insert a product into reserved list.
* void optimize ()
  Optimize to retrieve the best solution.
* auto calcPrice () const -> int
  Calculate price of the Wrapper(s)
* auto toString () const -> std::string
  Return a string represents Wrapper(s) of same type.

Private Attributes

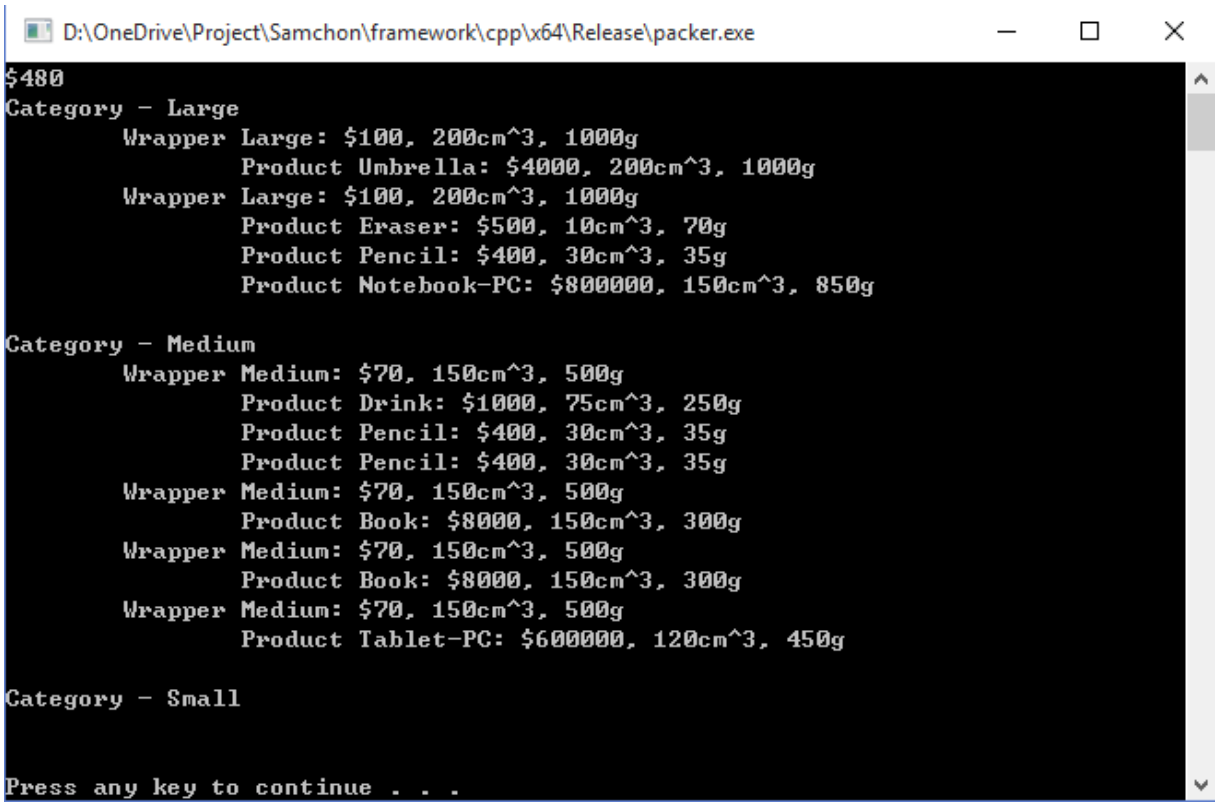
- std::vector< Product * > reserved
  A list for reserved Product(s).
- Wrapper * sample
  A sample wrapper used to copy.

6.144.1 Detailed Description

An array of wrapper with same category (name).

WrapperArray reserves Product(s) and generates enough number of Wrapper(s).

WrapperArray will retrieve the best arrange and packaging method for reserved Product(s).



Author

Jeongho Nam

6.144.2 Constructor & Destructor Documentation

6.144.2.1 WrapperArray (Wrapper * *sample*)

Construct from a sample wrapper.

Parameters

sample	A sample wrapper used to copy wrappers.

6.144.3 Member Function Documentation

6.144.3.1 auto tryInsert (Product * *product*) -> bool

Try to insert a product into reserved list.

If the Product's volume and weight is equal or less than the Wrapper categorized so that enable to insert in a Wrapper, reserve the Product and returns *true*. If not, does not reserve and just return *false*.

Returns

Whether the Product's volume and weight is equal or less than the Wrapper.

6.144.3.2 void optimize ( )

Optimize to retrieve the best solution.

Retrieve the best solution of packaging in level of WrapperArray.

Shuffles sequence of reserved Product(s) by samchon::library::FactorialGenerator and insert the reserved Products(s) following the sequence creating Wrapper(s) as needed. Between the sequences from FactorialGenerator, retrieve and determine the best solution.

Note

Sequence of inserting Product can affeact to numbers of Wrapper(s) to be used.

It's the reason why even WrapperArray has the optimize() method.

See also

samchon::library::FactorialGenerator

6.144.3.3 auto calcPrice ( ) const -> int

Calculate price of the Wrapper(s)

Calculate price of all wrapppers'. The price does not contain inserted products'

The documentation for this class was generated from the following files:

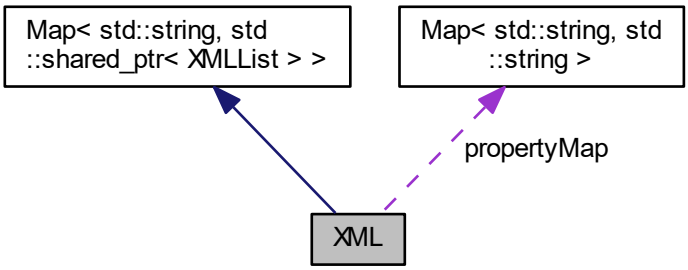
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/WrapperArray.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/examples/packer/WrapperArray.cpp

6.145 XML Class Reference

XML is a class representing xml object.

`#include <XML.hpp>`

Collaboration diagram for XML:



Public Member Functions

* XML ()
  Default Constructor.
* XML (const XML &)
  Copy Constructor.

* XML (XML &&)
  Move Constructor.
* XML (WeakString)
  Constructor by string.
* void push_back (const WeakString &)
  Add children xml objects by string representing them.
* void push_back (const std::shared_ptr< XML >)
  Add children xml.
* void addAllProperty (const std::shared_ptr< XML >)
  Add all properties from another XML.
* void setTag (const std::string &)
  Set tag (identifier) of the XML.
* template<typename _Ty >
  void setValue (const _Ty &val)
  Set value of the XML.
* template<typename _Ty >
  void setProperty (const std::string &name, const _Ty &val)
  Set a property with its key.
* void eraseProperty (const std::string &)
  Erase a property by its key.
* void clearProperties ()
  Remove all properties in the XML.
* auto getTag () const -> std::string
  Get key; identifer of the XML.
* template<class _Ty = std::string>
  auto getValue () const -> _Ty
  Get value of the XML.
* template<class _Ty = std::string>
  auto getProperty (const std::string &key) const -> _Ty
  Get property.
* auto hasProperty (const std::string &) const -> bool
  Test wheter a property exists or not.
* auto getPropertyMap () const -> const Map< std::string, std::string > &
  Get propertyMap.
* auto toString (size_t level=0) const -> std::string
  Get the string content.

Private Member Functions

* XML (XML *, WeakString &)
  Protected Constructor by string for child.

Private Attributes

* std::string tag
  Tag name.
* std::string value
  Value of the XML.
* Map< std::string, std::string > propertyMap
  Properties belongs to the XML.

6.145.1 Detailed Description

XML is a class representing xml object.

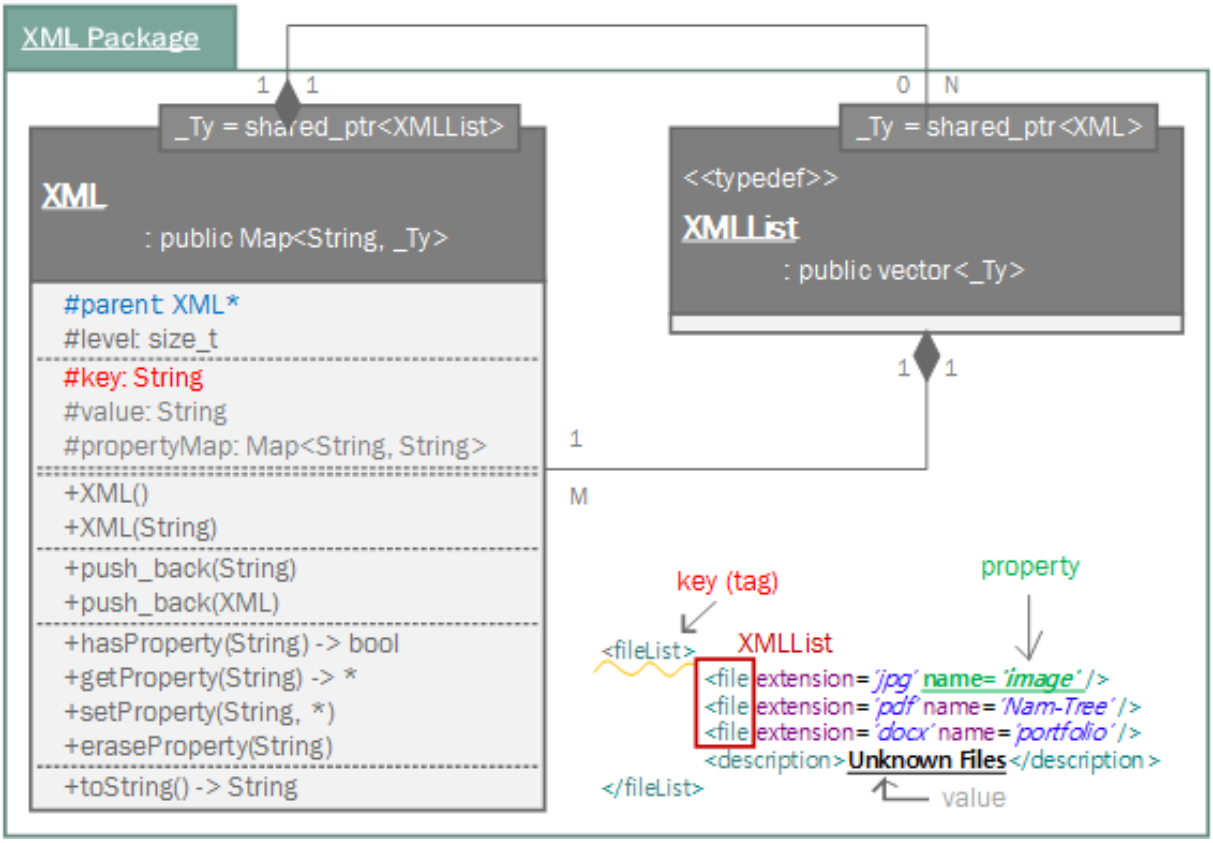
The XML class provides methods and properties for working with XML objects.

The XML class (along with the XMLList and Namespace) implements the powerful XML-handling standard defined in ECMAScript for XML (E4X) specification.

XML class has a recursive, hierarchical relationship.

All XML objects're managed by shared_ptr.

* XML contains XMLList from dictionary of shared pointer<XMLList>
* XMLList contains XML from vector of shared pointer<XML>
* Even if user creates an XML object directly, it's the basic principle to use shared pointer



Note

Parsing comment is not supported yet.

It's not recommeded to creating an XML object which is not being managed by shared pointer.

Warning

Do not abuse values for expressing member variables.

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

```
#include <iostream>
#include <vector>
#include <string>

#include <samchon/library/XML.hpp>

#ifdef _WIN64
#   ifdef _DEBUG
#       pragma comment(lib, "x64/Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "x64/Release/SamchonFramework.lib")
#   endif
#else
#   ifdef _DEBUG
#       pragma comment(lib, "Debug/SamchonFramework.lib")
#   else
#       pragma comment(lib, "Release/SamchonFramework.lib")
#   endif
#endif

```

```
using namespace std;
using namespace samchon::library;

struct Member
{
    string id;
    string name;
    int age;
    int grade;

    auto toXML() const -> shared_ptr<XML>
    {
        shared_ptr<XML> xml(new XML());
        xml->setTag("member");
        xml->setProperty("id", id);
        xml->setProperty("name", name);
        xml->setProperty("age", age);
        xml->setProperty("grade", grade);

        return xml;
    };
};

void main()
{
    //DECLARE An XML WITH MEMBER_ARRAY TAG
    shared_ptr<XML> xml(new XML());
    xml->setTag("memberArray");

    //ADD MEMBER(S) TO XML
    vector<struct Member> memberArray =
    {
        {"samchon", "Jeongho Nam", 27, 1},
        {"gkyu", "Kwangkyu Ko", 25, 1},
        {"guest", "John Doe", 99, 4}
    };
    for(size_t i = 0; i < memberArray.size(); i++)
        xml->push_back( memberArray[i].toXML() );

    //ADD FILE_LIST BY STRING DIRECTLY
    xml->push_back
    (
        string("") +
        "<fileList>\n" +
            "<file extension='pdf' name='API' />" +
            "<file extension='pdf' name='Guidance+For+Developer'>Damaged</file>" +
            "<file extension='docx' name='Resume' />" +
            "<file extension='jpg' name='My+House' />" +
            "<file extension='xlsx' name='Grades' />" +
        "</fileList>"
    );

    cout << "----------------------------------------------------------------" << endl;
    cout << " Get Properties And Values" << endl;
    cout << "----------------------------------------------------------------" << endl;
    cout << "Age of 2nd member: " << xml->get("member")->at(1)->getProperty<int>("age") << endl;
    cout << "Age of 1st member: " << xml->get("member")->at(0)->getProperty("id") << endl << endl;

    cout << "File name and extension of 5th: "
        << xml->get("fileList")->at(0)->get("file")->at(4)->getProperty("name") << "."
        << xml->get("fileList")->at(0)->get("file")->at(4)->getProperty("extension") << endl;
    cout << "Value of 2nd file: " << xml->get("fileList")->at(0)->get("file")->at(1)->getValue() << endl << endl;

    cout << "----------------------------------------------------------------" << endl;
    cout << " XML to String: " << endl;
    cout << "----------------------------------------------------------------" << endl;
    cout << xml->toString() << endl;

    system("pause");
}
```

See also

samchon::library

Author

Jeongho Nam

6.145.2 Constructor & Destructor Documentation

6.145.2.1 XML ( )

Default Constructor.

Warning

Declare XML to managed by shared pointer

6.145.2.2 XML (const XML & *xml*)

Copy Constructor.
Not copying (shared) pointer of children xml objects, but copying the real objects of children xml

6.145.2.3 XML (WeakString *wStr*)

Constructor by string.
Parses a string so that constructs an XML object
Parameters

str	A string representing xml object

Warning

Declare XML to managed by shared pointer

6.145.2.4 XML (XML * *parent*, WeakString & *str*) [private]

Protected Constructor by string for child.
Parses a string so that creates an XML object It is called for creating children XML objects from parent XML object.
Parameters

parent	Parent object who will contains this XML object
str	A string to be parsed

6.145.3 Member Function Documentation

6.145.3.1 void push_back (const WeakString & *str*)

Add children xml objects by string representing them.
Parameters

str	A string representing xml objects whould be belonged to this XML

6.145.3.2 void push_back (const std::shared_ptr< XML >)

Add children xml.
Parameters

xml	An xml object you want to add

6.145.3.3 void addAllProperty (const std::shared_ptr< XML >)

Add all properties from another XML.
Copies all properties from target to here.

Warning

Not a category of assign, but an insert.

Parameters

xml	Target xml object to deliver its properties

6.145.3.4 void setTag (const std::string & *tag*)

Set tag (identifier) of the XML.

See also

XML::tag

6.145.3.5 void setValue (const _Ty & *val*) [inline]

Set value of the XML.

Template Parameters

_Ty	Type of the value

Parameters

val	The value to set

Warning

Do not abuse values for expressing member variables

Standard Usage	Non-standard usage abusing value
<memberList> <member id='jhnam88' name='Jeongho+Nam' birthdate='1988-03-11' /> <member id='master' name='Administartor' birthdate='2011-07-28' /> </memberList>	<member> <id>jhnam88</id> <name>Jeongho+Nam</name> <birthdate>1988-03-11</birthdate> </member>

6.145.3.6 void eraseProperty (const std::string & *tag*)

Erase a property by its key.

Parameters

key	The key of the property to erase

Exceptions

exception	Unable to find the element

6.145.3.7 auto getTag ( ) const -> std::string

Get key; identifer of the XML.

Returns

tag, identifer of the XML

See also

XML::tag

6.145.3.8 auto toString (size_t *level* = 0) const -> std::string

Get the string content.

Returns a string representation of the XML and its all children

Returns

A string representing the xml

6.145.4 Member Data Documentation

6.145.4.1 std::string tag [private]

Tag name.

* <**tag** label='property' />: tag => "tag"
* <**price** high='1500' low='1300' open='1450' close='1320' />: tag => "price"

6.145.4.2 std::string value [private]

Value of the XML.

* <parameter name='age' type='int'>**26**</parameter>: value => 26
* <price high='1500' low='1300' open='1450' close='1320' />: tag => null

6.145.4.3 Map<std::string, std::string> propertyMap [private]

Properties belongs to the XML.

A Dictionary of properties accessing each property by its key.

* <price **high='1500' low='1300' open='1450' close='1320'** />: propertyMap => {{"high": 1500}, {"low": 1300}, {"open": 1450}, {"close", 1320}}
* <member **id='jhnam88' name='Jeongho+Nam' comment='Hello.+My+name+is+Jeongho+Nam'** >: propertyMap => {{"id", "jhnam88"}, {"name", "Jeongho Nam"}, {"comment", "Hello. My name is Jeongho Nam"}}

The documentation for this class was generated from the following files:

* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/XML.hpp
* D:/OneDrive/Project/Samchon/framework/cpp/samchon/library/XML.cpp

Index

_replyData
samchon::protocol::IClient, 284
samchon::protocol::master::ParallelSystem, 412
samchon::protocol::slave::ParallelSystem, 416
samchon::protocol::slave::SlaveSystem, 462
~ChatClient
samchon::example::console_chat_server::ChatClient, 85
~SmartPointer
samchon::SmartPointer, 464
~UniqueAcquire
samchon::library::UniqueAcquire, 490
~UniqueReadLock
samchon::library::UniqueReadLock, 493
~UniqueWriteLock
samchon::library::UniqueWriteLock, 496

accept
samchon::protocol::FlashPolicyServer, 249
acquire
samchon::library::Semaphore, 439
samchon::library::SharedAcquire, 449
samchon::library::UniqueAcquire, 490
addAllProperty
samchon::library::XML, 524
addClient
samchon::protocol::service::User, 501
addDate
samchon::library::Date, 117
samchon::library::Datetime, 126
addEventListener
samchon::library::EventDispatcher, 222
addHour
samchon::library::Datetime, 126
addMinute
samchon::library::Datetime, 126
addMonth
samchon::library::Date, 117
samchon::library::Datetime, 124
addSecond
samchon::library::Datetime, 127
addTab
samchon::library::StringUtil, 478
addWeek
samchon::library::Date, 117
samchon::library::Datetime, 126
addYear
samchon::library::Date, 117
samchon::library::Datetime, 124
allocate
samchon::library::CriticalAllocator, 110
allocateRoles
samchon::protocol::master::DistributedSystemArray, 169
application
samchon::library::FTFactory, 253
archive
samchon::protocol::ISQLEntity, 328
at
samchon::WeakString, 509
samchon::library::CaseGenerator, 78
samchon::library::SQLStatement, 471
authority
samchon::protocol::service::User, 503
avgElapsedTime
samchon::protocol::master::DistributedSystem, 165

Base64, 63
begin
samchon::namtree::NTEntityGroup, 353
samchon::protocol::service::Server, 441
between
samchon::WeakString, 511
samchon::library::StringUtil, 478
betweens
samchon::WeakString, 511
samchon::library::StringUtil, 479
bindParameterBASizeMap
samchon::library::SQLStatement, 472
ByteArray, 65

CHILD_TAG
samchon::example::chat_service::ChatRoomArray, 93
samchon::library::FTFolder, 258
samchon::namtree::NTCriteria, 351
samchon::namtree::NTParameter, 365
samchon::namtree::NTParameterArray, 368
samchon::protocol::ExternalSystem, 238
samchon::protocol::ExternalSystemArray, 242
samchon::protocol::IEntityGroup, 287
samchon::protocol::Invoke, 305
samchon::protocol::InvokeHistoryArray, 311
calcDistance
samchon::example::tsp::GeometryPoint, 280
calcElapsedTime
samchon::protocol::InvokeHistory, 308
calcLastDates
samchon::library::Date, 115
calcPrice
samchon::example::packer::WrapperArray, 520
calcSeconds
samchon::library::Date, 115

CaseGenerator, 69
samchon::library::CaseGenerator, 78
category
samchon::library::FTFactory, 253
Charset, 79
ChatClient, 82, 84
samchon::example::console_chat_server::ChatClient, 85
ChatMessage, 86
ChatRoom, 88
samchon::example::chat_service::ChatRoom, 90
ChatRoomArray, 91
ChatServer, 94, 95
ChatService, 97
ChatUser, 98
Client, 100
samchon::protocol::service::Client, 102
colorNumberFormat
samchon::library::StringUtil, 476
colorPercentFormat
samchon::library::StringUtil, 476
CombinedPermutationGenerator, 105
samchon::library::CombinedPermutationGenerator, 106
compress
samchon::ByteArray, 68
connect
samchon::library::SQLi, 467
samchon::protocol::ServerConnector, 443
construct
samchon::example::chat_service::ChatMessage, 87
samchon::library::CriticalAllocator, 109
samchon::namtree::INTExplore, 299
samchon::namtree::NTCriteria, 351
samchon::namtree::NTFile, 360
samchon::namtree::NTParameter, 365
samchon::namtree::NTParameterDetermined, 371
samchon::namtree::NTSide, 374
samchon::protocol::Entity, 188
samchon::protocol::EntityArray, 194
samchon::protocol::EntityGroup, 203
samchon::protocol::EntityList, 212
samchon::protocol::ExternalClientArray, 228
samchon::protocol::ExternalServer, 231
samchon::protocol::ExternalSystem, 237
samchon::protocol::ExternalSystemRole, 245
samchon::protocol::Invoke, 304
samchon::protocol::InvokeHistory, 308
samchon::protocol::InvokeParameter, 315
samchon::protocol::PRInvokeHistory, 429
samchon::protocol::SystemRole, 481
samchon::protocol::master::DistributedClientArray, 135
samchon::protocol::master::DistributedClientArrayMediator, 138
samchon::protocol::master::DistributedServer, 145
samchon::protocol::master::DistributedSystem, 163
samchon::protocol::master::DistributedSystemArray, 168
samchon::protocol::master::DistributedSystemArrayMediator, 174
samchon::protocol::master::ParallelClientArrayMediator, 387
samchon::protocol::master::ParallelServer, 392
samchon::protocol::master::ParallelSystem, 412
samchon::protocol::master::ParallelSystemArrayMediator
constructService
samchon::protocol::service::Client, 105
createChild
samchon::namtree::NTCriteria, 351
samchon::namtree::NTParameter, 365
samchon::namtree::NTParameterArray, 368
samchon::protocol::EntityGroup, 204
samchon::protocol::Invoke, 304
samchon::protocol::InvokeHistoryArray, 311
samchon::protocol::master::DSInvokeHistoryList, 182
samchon::protocol::master::PRInvokeHistoryArray, 430
createCriteria
samchon::namtree::NTFactory, 356
createRole
samchon::protocol::master::DistributedSystemArray, 169
createRoom
samchon::example::chat_service::ListService, 336
createService
samchon::example::chat_service::ChatClient, 83
samchon::protocol::service::Client, 103
createSide
samchon::namtree::NTFactory, 356
createSlave
samchon::protocol::master::DistributedSystemArrayMedia
samchon::protocol::master::ParallelSystemArrayMediator
createStatement
samchon::library::SQLi, 467
samchon::library::TSQLi, 486
CriticalAllocator< _Ty >, 107
CriticalDictionary
samchon::library, 29
CriticalList
samchon::library, 29
CriticalMap
samchon::library, 30
CriticalSet
samchon::library, 30
CriticalVector
samchon::library, 30
crossover
samchon::library::GeneticAlgorithm, 277

DSInvokeHistory, 179
samchon::protocol::master::DSInvokeHistory, 181
DSInvokeHistoryList, 181
data
samchon::WeakString, 508
Date, 111
samchon::library::Date, 114, 115
Datetime, 119
samchon::library::Datetime, 121, 122
deallocate
samchon::library::CriticalAllocator, 111
decode
samchon::library::Base64, 64
samchon::library::URLVariables, 498
decompress
samchon::ByteArray, 68

denominator
samchon::library::ProgressEvent, 434
destroy
samchon::library::CriticalAllocator, 110
Dictionary
samchon, 16
dispatchEvent
samchon::library::EventDispatcher, 222
dispatchProgressEvent
samchon::library::EventDispatcher, 223
distance
samchon::example::tsp::Travel, 484
DistributedClient, 128
DistributedClientArray, 130
DistributedClientArrayMediator, 135
DistributedClientServerMediator, 140
DistributedServer, 143
DistributedServerArray, 146
DistributedServerArrayMediator, 149
DistributedSlaveServerMediator, 153
DistributedSlaveSystemMediator, 157
DistributedSystem, 160
DistributedSystemArray, 165
DistributedSystemArrayMediator, 170
DistributedSystemRole, 176
doJoin
samchon::protocol::service::User, 502
doLogin
samchon::protocol::service::User, 502
driver
samchon::library::SQLi, 467

E
samchon::library::Math, 344
empty
samchon::WeakString, 508
encode
samchon::library::Base64, 64
samchon::library::URLVariables, 498
end
samchon::namtree::NTEntityGroup, 353
samchon::protocol::service::Server, 442
endTime
samchon::protocol::InvokeHistory, 310
Entity, 183
EntityArray< _Ty >, 189
EntityGroup< _Container, _ETy, _Ty >, 195
EntityList< _Ty >, 206
eraseClient
samchon::example::chat_service::ChatRoom, 91
samchon::protocol::service::User, 501
eraseProperty
samchon::library::XML, 525
ErrorEvent, 213
samchon::library::ErrorEvent, 216
Event, 216
samchon::library::Event, 219
EventDispatcher, 219
samchon::library::EventDispatcher, 222
evolveGeneArray
samchon::library::GeneticAlgorithm, 276
evolvePopulation
samchon::library::GeneticAlgorithm, 276
execute
samchon::library::SQLStatement, 470
executeDirectly
samchon::library::SQLStatement, 470
extension
samchon::library::FTFile, 256
ExternalClient, 223
ExternalClientArray, 225
ExternalServer, 229
ExternalServerArray, 232
ExternalSystem, 235
ExternalSystemArray, 239
ExternalSystemRole, 242
samchon::protocol::ExternalSystemRole, 244

FTByteFile, 250
samchon::library::FTByteFile, 251
FTFactory, 251
FTFile, 254
samchon::library::FTFile, 255
FTFolder, 256
samchon::library::FTFolder, 258
FTInstance, 259
samchon::library::FTInstance, 261
FTTextFile, 262
samchon::library::FTTextFile, 264
FactorialGenerator, 247
samchon::library::FactorialGenerator, 248
factory
samchon::namtree::NTFile, 360
fetch
samchon::library::SQLStatement, 471
find
samchon::WeakString, 509
finds
samchon::WeakString, 509
samchon::library::StringUtil, 477
fitTest
samchon::library::GAPopulation, 269
FlashPolicyServer, 248
samchon::protocol::FlashPolicyServer, 249

GAParameters, 264, 265
GAPopulation
samchon::library::GAPopulation, 268
GAPopulation< GeneArray, Compare >, 266
GeneticAlgorithm
samchon::library::GeneticAlgorithm, 276
GeneticAlgorithm< GeneArray, Compare >, 269
GeometryPoint, 278
samchon::example::tsp::GeometryPoint, 279
get
samchon::Map, 338
samchon::SmartPointer, 464
samchon::library::SQLStatement, 471

samchon::protocol::EntityGroup, 204
getAuthority
samchon::protocol::service::User, 501
getDate
samchon::library::Date, 118
getDay
samchon::library::Date, 118
getHour
samchon::library::Datetime, 127
getIndex
samchon::IndexPair, 294
getListener
samchon::example::chat_service::ChatMessage, 88
getMinute
samchon::library::Datetime, 127
getMonth
samchon::library::Date, 118
getPosition
samchon::ByteArray, 66
getRole
samchon::protocol::ExternalSystemArray, 241
samchon::protocol::master::DistributedSystemArray, 169
getSecond
samchon::library::Datetime, 127
getSemaphore
samchon::protocol::service::User, 501
getSessionID
samchon::protocol::service::IPUserPair, 322
getSystem
samchon::protocol::master::DistributedSystemRole, 178
getTag
samchon::library::XML, 525
getValue
samchon::IndexPair, 294, 295
samchon::protocol::InvokeParameter, 315
getYear
samchon::library::Date, 118
getvalueAsXML
samchon::protocol::InvokeParameter, 316
goJoin
samchon::protocol::service::User, 502
goLogin
samchon::protocol::service::User, 502
goLogout
samchon::protocol::service::User, 502

HTTPLoader, 281
has
samchon::Map, 338
samchon::protocol::EntityGroup, 204
hasRole
samchon::protocol::ExternalSystemArray, 241

IClient, 282
IEntityChain, 285
samchon::protocol::IEntityChain, 286
IEntityGroup, 287
IEntityPtrGroup< _Container >, 288
IHTMLEntity, 288
INTExplore, 296
IOperator< _Ty >, 317
IPUserPair, 320
IProtocol, 318
ISQLEntity, 327
IServer, 322
IWebClient, 328
IWebServer, 331
id
samchon::protocol::service::User, 503
inProgress
samchon::protocol::master::DistributedSystem, 165
IndexPair< _Ty >, 294
Instance, 295
samchon::example::packer::Instance, 296
instanceMap
samchon::library::FTFactory, 253
Invoke, 300
samchon::protocol::Invoke, 303
InvokeHistory, 306
samchon::protocol::InvokeHistory, 308
InvokeHistoryArray, 310
InvokeParameter, 312
samchon::protocol::InvokeParameter, 314, 315
isNumeric
samchon::library::StringUtil, 475
issueSessionID
samchon::protocol::service::IPUserPair, 322

key
samchon::library::FTInstance, 261
samchon::namtree::NTParameter, 365
samchon::namtree::NTParameterDetermined, 371
samchon::protocol::Entity, 188
samchon::protocol::ExternalSystem, 237
samchon::protocol::ExternalSystemRole, 245
samchon::protocol::InvokeHistory, 308
samchon::protocol::InvokeParameter, 315
samchon::protocol::SystemRole, 481

LN10
samchon::library::Math, 345
LN2
samchon::library::Math, 345
LOG10E
samchon::library::Math, 345
LOG2E
samchon::library::Math, 345
label
samchon::namtree::NTParameterDetermined, 372
ListService, 335
listen
samchon::protocol::IClient, 284
samchon::protocol::IWebClient, 330
listener
samchon::example::chat_service::ChatMessage, 88
samchon::protocol::InvokeHistory, 309
load
samchon::protocol::ISQLEntity, 328

localEndPoint
samchon::protocol::ServerConnector, 444
lock
samchon::library::SharedReadLock, 453
samchon::library::SharedWriteLock, 455
samchon::library::UniqueReadLock, 493
samchon::library::UniqueWriteLock, 496
ltrim
samchon::WeakString, 512
samchon::library::StringUtil, 477

Map< _Kty, _Ty, _Pr, _Alloc >, 336
Math, 339
maximum
samchon::library::Math, 342
mean
samchon::library::Math, 342
median
samchon::library::Math, 342
minimum
samchon::library::Math, 341
mode
samchon::library::Math, 343
moveValue
samchon::protocol::InvokeParameter, 316
mtx
samchon::library::Semaphore, 439
mutate
samchon::library::GeneticAlgorithm, 277
mutationRate
samchon::library::GAParameters, 265
samchon::library::GeneticAlgorithm, 278
MySQLi, 346
samchon::library::MySQLi, 347

NTCriteria, 348
NTEntityGroup, 352
NTFactory, 354
samchon::namtree::NTFactory, 356
NTFile, 357
samchon::namtree::NTFile, 359
NTIterator, 361
NTParameter, 363
NTParameterArray, 366
NTParameterDetermined, 369
NTSide, 372
name
samchon::example::chat_service::ChatRoom, 91
samchon::namtree::NTParameter, 366
samchon::protocol::ExternalSystem, 239
next
samchon::library::SQLStatement, 471
notify
samchon::example::chat_service::ChatRoomArray, 93
notifyEnd
samchon::protocol::InvokeHistory, 308
npos
samchon::WeakString, 513
numberFormat
samchon::library::StringUtil, 475

open
samchon::protocol::IServer, 326
samchon::protocol::IWebServer, 334
operator<
samchon::example::tsp::Travel, 483
samchon::library::IOperator, 318
samchon::namtree::NTIterator, 363
operator*
samchon::SmartPointer, 464
operator++
samchon::namtree::NTIterator, 362
operator->
samchon::SmartPointer, 464
operator--
samchon::namtree::NTIterator, 362
operator==
samchon::library::IOperator, 318
samchon::namtree::NTIterator, 362
operator[]
samchon::WeakString, 509
samchon::library::CaseGenerator, 78
samchon::library::CombinedPermutationGenerator, 106
samchon::library::PermutationGenerator, 426
optimize
samchon::example::packer::WrapperArray, 520
samchon::example::tsp::Scheduler, 437
otherside
samchon::namtree::NTFile, 360

PI
samchon::library::Math, 344
PRInvokeHistory, 427
samchon::protocol::PRInvokeHistory, 428
PRInvokeHistoryArray, 429
Packer, 375
samchon::example::packer::Packer, 376
ParallelClient, 377, 379
ParallelClientArray, 380
ParallelClientArrayMediator, 383
ParallelServer, 389, 391
ParallelServerArray, 393
ParallelServerArrayMediator, 396
ParallelSlaveClientMediator, 400
ParallelSlaveServerMediator, 404
ParallelSlaveSystemMediator, 407
ParallelSystem, 410, 414
ParallelSystemArray, 417
ParallelSystemArrayMediator, 419
percentFormat
samchon::library::StringUtil, 476
performance
samchon::protocol::master::DistributedSystem, 165
samchon::protocol::master::ParallelSystem, 414
PermutationGenerator, 424
samchon::library::PermutationGenerator, 426
pop
samchon::Map, 339

port
samchon::library::SQLi, 467
position
samchon::ByteArray, 68
prepare
samchon::library::SQLStatement, 470
Product, 431
samchon::example::packer::Product, 432
ProgressEvent, 432
samchon::library::ProgressEvent, 433
propertyMap
samchon::library::XML, 526
push_back
samchon::library::XML, 524

RWMutex, 434
random
samchon::library::Math, 341
read
samchon::ByteArray, 67
readLock
samchon::library::RWMutex, 435
readReversely
samchon::ByteArray, 67
readUnlock
samchon::library::RWMutex, 435
referValue
samchon::protocol::InvokeParameter, 316
registerClient
samchon::example::chat_service::ChatRoom, 90
release
samchon::library::Semaphore, 439
samchon::library::SharedAcquire, 449
samchon::library::UniqueAcquire, 490
removeEventListener
samchon::library::EventDispatcher, 222
replace
samchon::WeakString, 512
replaceAll
samchon::WeakString, 512, 513
samchon::library::StringUtil, 480
replyData
samchon::example::console_chat_server::ChatClient, 85
samchon::example::console_chat_server::ChatServer, 95
samchon::protocol::ExternalSystem, 237
samchon::protocol::ExternalSystemArray, 241
samchon::protocol::IProtocol, 319
samchon::protocol::master::DistributedSlaveSystemMediator, 160
samchon::protocol::master::DistributedSystem, 164
samchon::protocol::master::ParallelSlaveSystemMediator, 412
samchon::protocol::service::Client, 103
samchon::protocol::service::User, 503
samchon::protocol::slave::ParallelSystem, 417
reset
samchon::SmartPointer, 464
reverse
samchon::ByteArray, 66
rfind
samchon::WeakString, 509
rfinds
samchon::WeakString, 510
samchon::library::StringUtil, 477
room
samchon::example::chat_service::ChatService, 98
rtrim
samchon::WeakString, 512
samchon::library::StringUtil, 477

SQLStatement, 467
samchon::library::SQLStatement, 470
SQLi, 465
samchon::library::SQLi, 466
SQRT1_2
samchon::library::Math, 345
SQRT2
samchon::library::Math, 345
samchon, 15
Dictionary, 16
samchon::ByteArray
compress, 68
decompress, 68
getPosition, 66
position, 68
read, 67
readReversely, 67
reverse, 66
setPosition, 66
write, 67
writeReversely, 68
samchon::IndexPair
getIndex, 294
getValue, 294, 295
samchon::Map
get, 338
has, 338
pop, 339
set, 339
samchon::SmartPointer
~SmartPointer, 464
get, 464
operator*, 464
operator->, 464
reset, 464
SmartPointer, 463
samchon::WeakString
at, 509
between, 511
betweens, 511
data, 508
empty, 508
find, 509
finds, 509
ltrim, 512
npos, 513
operator[], 509
replace, 512
replaceAll, 512, 513
rfind, 509

rfinds, 510
rtrim, 512
size, 508
split, 511
str, 513
substr, 510
substring, 510
toLowerCase, 513
toUpperCase, 513
trim, 512
WeakString, 507, 508
samchon::example::chat_service, 17
samchon::example::chat_service::ChatClient
createService, 83
samchon::example::chat_service::ChatMessage
construct, 87
getListener, 88
listener, 88
TAG, 87
toXML, 88
samchon::example::chat_service::ChatRoom
ChatRoom, 90
eraseClient, 91
name, 91
registerClient, 90
TAG, 90
toXML, 91
samchon::example::chat_service::ChatRoomArray
CHILD_TAG, 93
notify, 93
samchon::example::chat_service::ChatService
room, 98
samchon::example::chat_service::ListService
createRoom, 336
samchon::example::console_chat_server, 18
samchon::example::console_chat_server::ChatClient
~ChatClient, 85
ChatClient, 85
replyData, 85
samchon::example::console_chat_server::ChatServer
replyData, 95
sendData, 95
samchon::example::packer::Instance
Instance, 296
samchon::example::packer::Packer
Packer, 376
samchon::example::packer::Product
Product, 432
samchon::example::packer::Wrapper
toString, 518
tryInsert, 517
Wrapper, 517
samchon::example::packer::WrapperArray
calcPrice, 520
optimize, 520
tryInsert, 519
WrapperArray, 519
samchon::example::tsp, 19
samchon::example::tsp::GeometryPoint
calcDistance, 280
GeometryPoint, 279
toString, 281
samchon::example::tsp::Scheduler
optimize, 437
samchon::example::tsp::Travel
distance, 484
operator<, 483
toString, 483
Travel, 483
samchon::library, 22
CriticalDictionary, 29
CriticalList, 29
CriticalMap, 30
CriticalSet, 30
CriticalVector, 30
XMLList, 30
samchon::library::Base64
decode, 64
encode, 64
samchon::library::CaseGenerator
at, 78
CaseGenerator, 78
operator[], 78
size, 78
toMatrix, 78
samchon::library::Charset
toMultibyte, 80, 81
toUTF8, 81
toUnicode, 82
samchon::library::CombinedPermutationGenerator
CombinedPermutationGenerator, 106
operator[], 106
samchon::library::CriticalAllocator
allocate, 110
construct, 109
deallocate, 111
destroy, 110
samchon::library::Date
addDate, 117
addMonth, 117
addWeek, 117
addYear, 117
calcLastDates, 115
calcSeconds, 115
Date, 114, 115
getDate, 118
getDay, 118
getMonth, 118
getYear, 118
set, 115, 116
setDate, 117
setMonth, 116
setYear, 116
TP_1970, 119
toLinuxTime, 118
toString, 118

toTM, 118
samchon::library::Datetime
addDate, 126
addHour, 126
addMinute, 126
addMonth, 124
addSecond, 127
addWeek, 126
addYear, 124
Datetime, 121, 122
getHour, 127
getMinute, 127
getSecond, 127
set, 122, 123
setDate, 123
setHour, 124
setMinute, 124
setMonth, 123
setSecond, 124
setYear, 123
toString, 127
samchon::library::ErrorEvent
ErrorEvent, 216
samchon::library::Event
Event, 219
samchon::library::EventDispatcher
addEventListener, 222
dispatchEvent, 222
dispatchProgressEvent, 223
EventDispatcher, 222
removeEventListener, 222
samchon::library::FTByteFile
FTByteFile, 251
samchon::library::FTFactory
application, 253
category, 253
instanceMap, 253
samchon::library::FTFile
extension, 256
FTFile, 255
toXML, 255
samchon::library::FTFolder
CHILD_TAG, 258
FTFolder, 258
toXML, 258
samchon::library::FTInstance
FTInstance, 261
key, 261
TAG, 261
toXML, 262
samchon::library::FTTextFile
FTTextFile, 264
samchon::library::FactorialGenerator
FactorialGenerator, 248
samchon::library::GAParameters
mutationRate, 265
samchon::library::GAPopulation
fitTest, 269
GAPopulation, 268
samchon::library::GeneticAlgorithm
crossover, 277
evolveGeneArray, 276
evolvePopulation, 276
GeneticAlgorithm, 276
mutate, 277
mutationRate, 278
selection, 276
samchon::library::IOperator
operator<, 318
operator==, 318
samchon::library::Math
E, 344
LN10, 345
LN2, 345
LOG10E, 345
LOG2E, 345
maximum, 342
mean, 342
median, 342
minimum, 341
mode, 343
PI, 344
random, 341
SQRT1_2, 345
SQRT2, 345
stdev_p, 343
stdev_s, 343
variance_p, 344
variance_s, 344
samchon::library::MySQLi
MySQLi, 347
samchon::library::PermutationGenerator
operator[], 426
PermutationGenerator, 426
samchon::library::ProgressEvent
denominator, 434
ProgressEvent, 433
samchon::library::RWMutex
readLock, 435
readUnlock, 435
writeLock, 436
samchon::library::SQLStatement
at, 471
bindParameterBASizeMap, 472
execute, 470
executeDirectly, 470
fetch, 471
get, 471
next, 471
prepare, 470
SQLStatement, 470
size, 471
toXML, 471
samchon::library::SQLi
connect, 467
createStatement, 467

driver, 467
port, 467
SQLi, 466
samchon::library::Semaphore
acquire, 439
mtx, 439
release, 439
Semaphore, 438
setSize, 439
size, 439
size_, 439
tryAcquire, 439
samchon::library::SharedAcquire
acquire, 449
release, 449
SharedAcquire, 449
samchon::library::SharedReadLock
lock, 453
SharedReadLock, 452
unlock, 453
samchon::library::SharedWriteLock
lock, 455
SharedWriteLock, 455
unlock, 456
samchon::library::StringUtil
addTab, 478
between, 478
betweens, 479
colorNumberFormat, 476
colorPercentFormat, 476
finds, 477
isNumeric, 475
ltrim, 477
numberFormat, 475
percentFormat, 476
replaceAll, 480
rfinds, 477
rtrim, 477
split, 479
substitute, 474
substituteSQL, 475
substring, 478
toLowerCase, 479
toNumber, 475
toUpperCase, 479
trim, 476
samchon::library::TSQLStatement
toXML, 488
samchon::library::TSQLi
createStatement, 486
TSQLi, 486
samchon::library::URLVariables
decode, 498
encode, 498
toString, 498
URLVariables, 497
samchon::library::UniqueAcquire
~UniqueAcquire, 490
acquire, 490
release, 490
UniqueAcquire, 490
samchon::library::UniqueReadLock
~UniqueReadLock, 493
lock, 493
UniqueReadLock, 492, 493
unlock, 493
samchon::library::UniqueWriteLock
~UniqueWriteLock, 496
lock, 496
UniqueWriteLock, 495, 496
unlock, 496
samchon::library::XML
addAllProperty, 524
eraseProperty, 525
getTag, 525
propertyMap, 526
push_back, 524
setTag, 525
setValue, 525
tag, 526
toString, 525
value, 526
XML, 523, 524
samchon::namtree, 31
samchon::namtree::INTExplore
construct, 299
toXML, 299
samchon::namtree::NTCriteria
CHILD_TAG, 351
construct, 351
createChild, 351
TAG, 351
toXML, 351
samchon::namtree::NTEntityGroup
begin, 353
end, 353
samchon::namtree::NTFactory
createCriteria, 356
createSide, 356
NTFactory, 356
samchon::namtree::NTFile
construct, 360
factory, 360
NTFile, 359
otherside, 360
toXML, 360
samchon::namtree::NTIterator
operator<, 363
operator++, 362
operator--, 362
operator==, 362
samchon::namtree::NTParameter
CHILD_TAG, 365
construct, 365
createChild, 365
key, 365

name, 366
TAG, 365
toXML, 366
samchon::namtree::NTParameterArray
CHILD_TAG, 368
createChild, 368
TAG, 368
samchon::namtree::NTParameterDetermined
construct, 371
key, 371
label, 372
TAG, 371
toXML, 371
samchon::namtree::NTSide
construct, 374
TAG, 374
toXML, 374
samchon::protocol, 35
SharedEntityArray, 46
SharedEntityList, 47
UniqueEntityArray, 47
UniqueEntityList, 47
samchon::protocol::Entity
construct, 188
key, 188
TAG, 187
toXML, 188
samchon::protocol::EntityArray
construct, 194
toXML, 194
samchon::protocol::EntityGroup
construct, 203
createChild, 204
get, 204
has, 204
toXML, 206
samchon::protocol::EntityList
construct, 212
toXML, 212
samchon::protocol::ExternalClientArray
construct, 228
start, 228
toXML, 228
samchon::protocol::ExternalServer
construct, 231
start, 231
toXML, 231
samchon::protocol::ExternalServerArray
start, 234
samchon::protocol::ExternalSystem
CHILD_TAG, 238
construct, 237
key, 237
name, 239
replyData, 237
start, 237
TAG, 238
toXML, 238
samchon::protocol::ExternalSystemArray
CHILD_TAG, 242
getRole, 241
hasRole, 241
replyData, 241
sendData, 241
start, 240
TAG, 242
samchon::protocol::ExternalSystemRole
construct, 245
ExternalSystemRole, 244
key, 245
sendData, 245
TAG, 245
toXML, 245
samchon::protocol::FlashPolicyServer
accept, 249
FlashPolicyServer, 249
samchon::protocol::IClient
_replyData, 284
listen, 284
sendData, 284
samchon::protocol::IEntityChain
IEntityChain, 286
samchon::protocol::IEntityGroup
CHILD_TAG, 287
samchon::protocol::IHTMLEntity
toHTML, 293
toTH, 292
toTR, 293
samchon::protocol::IProtocol
replyData, 319
sendData, 319
samchon::protocol::ISQLEntity
archive, 328
load, 328
toSQL, 328
samchon::protocol::IServer
open, 326
samchon::protocol::IWebClient
listen, 330
sendData, 330
samchon::protocol::IWebServer
open, 334
samchon::protocol::Invoke
CHILD_TAG, 305
construct, 304
createChild, 304
Invoke, 303
TAG, 304
toXML, 305
samchon::protocol::InvokeHistory
calcElapsedTime, 308
construct, 308
endTime, 310
InvokeHistory, 308
key, 308
listener, 309

notifyEnd, 308
startTime, 309
TAG, 308
toInvoke, 309
toXML, 309
samchon::protocol::InvokeHistoryArray
CHILD_TAG, 311
createChild, 311
TAG, 311
samchon::protocol::InvokeParameter
construct, 315
getValue, 315
getvalueAsXML, 316
InvokeParameter, 314, 315
key, 315
moveValue, 316
referValue, 316
TAG, 316
toXML, 316
type, 317
samchon::protocol::PRInvokeHistory
construct, 429
PRInvokeHistory, 428
toXML, 429
samchon::protocol::ServerConnector
connect, 443
localEndPoint, 444
samchon::protocol::SystemRole
construct, 481
key, 481
TAG, 481
toXML, 481
samchon::protocol::master, 48
samchon::protocol::master::DSInvokeHistory
DSInvokeHistory, 181
samchon::protocol::master::DSInvokeHistoryList
createChild, 182
samchon::protocol::master::DistributedClientArray
construct, 133
start, 134
toXML, 134
samchon::protocol::master::DistributedClientArrayMediator
construct, 138
start, 139
toXML, 139
samchon::protocol::master::DistributedServer
construct, 145
toXML, 146
samchon::protocol::master::DistributedServerArray
start, 149
samchon::protocol::master::DistributedServerArrayMediator
start, 153
samchon::protocol::master::DistributedSlaveSystemMediator
replyData, 160
TAG, 160
samchon::protocol::master::DistributedSystem
avgElapsedTime, 165
construct, 163
inProgress, 165
performance, 165
replyData, 164
sendData, 164
toXML, 164
samchon::protocol::master::DistributedSystemArray
allocateRoles, 169
construct, 168
createRole, 169
getRole, 169
start, 169
toXML, 169
samchon::protocol::master::DistributedSystemArrayMediator
construct, 174
createSlave, 175
start, 175
toXML, 175
samchon::protocol::master::DistributedSystemRole
getSystem, 178
sendData, 178
toXML, 178
samchon::protocol::master::PRInvokeHistoryArray
createChild, 430
samchon::protocol::master::ParallelClientArrayMediator
construct, 387
start, 388
toXML, 388
samchon::protocol::master::ParallelServer
construct, 392
toXML, 393
samchon::protocol::master::ParallelServerArrayMediator
start, 400
samchon::protocol::master::ParallelSlaveSystemMediator
replyData, 410
TAG, 410
samchon::protocol::master::ParallelSystem
_replyData, 412
construct, 412
performance, 414
sendSegmentData, 413
toXML, 413
samchon::protocol::master::ParallelSystemArray
sendSegmentData, 419
samchon::protocol::master::ParallelSystemArrayMediator
construct, 423
createSlave, 423
start, 424
toXML, 424
samchon::protocol::service, 51
samchon::protocol::service::Client
Client, 102
constructService, 105
createService, 103
replyData, 103
sendData, 103
samchon::protocol::service::IPUserPair
getSessionID, 322
issueSessionID, 322

samchon::protocol::service::Server
begin, 441
end, 442
samchon::protocol::service::Service
sendData, 445
Service, 445
samchon::protocol::service::ServiceKeeper
ServiceKeeper, 447
samchon::protocol::service::User
addClient, 501
authority, 503
doJoin, 502
doLogin, 502
eraseClient, 501
getAuthority, 501
getSemaphore, 501
goJoin, 502
goLogin, 502
goLogout, 502
id, 503
replyData, 503
semaphore, 503
sendData, 503
sessionID, 503
User, 500
samchon::protocol::slave, 61
samchon::protocol::slave::ParallelSystem
_replyData, 416
replyData, 417
samchon::protocol::slave::SlaveSystem
_replyData, 462
Scheduler, 436
selection
samchon::library::GeneticAlgorithm, 276
Semaphore, 437
samchon::library::Semaphore, 438
semaphore
samchon::protocol::service::User, 503
sendData
samchon::example::console_chat_server::ChatServer, 95
samchon::protocol::ExternalSystemArray, 241
samchon::protocol::ExternalSystemRole, 245
samchon::protocol::IClient, 284
samchon::protocol::IProtocol, 319
samchon::protocol::IWebClient, 330
samchon::protocol::master::DistributedSystem, 164
samchon::protocol::master::DistributedSystemRole, 178
samchon::protocol::service::Client, 103
samchon::protocol::service::Service, 445
samchon::protocol::service::User, 503
sendSegmentData
samchon::protocol::master::ParallelSystem, 413
samchon::protocol::master::ParallelSystemArray, 419
Server, 440
ServerConnector, 442
Service, 444
samchon::protocol::service::Service, 445
ServiceKeeper, 446
samchon::protocol::service::ServiceKeeper, 447
sessionID
samchon::protocol::service::User, 503
set
samchon::Map, 339
samchon::library::Date, 115, 116
samchon::library::Datetime, 122, 123
setDate
samchon::library::Date, 117
samchon::library::Datetime, 123
setHour
samchon::library::Datetime, 124
setMinute
samchon::library::Datetime, 124
setMonth
samchon::library::Date, 116
samchon::library::Datetime, 123
setPosition
samchon::ByteArray, 66
setSecond
samchon::library::Datetime, 124
setSize
samchon::library::Semaphore, 439
setTag
samchon::library::XML, 525
setValue
samchon::library::XML, 525
setYear
samchon::library::Date, 116
samchon::library::Datetime, 123
SharedAcquire, 447
samchon::library::SharedAcquire, 449
SharedEntityArray
samchon::protocol, 46
SharedEntityList
samchon::protocol, 47
SharedReadLock, 450
samchon::library::SharedReadLock, 452
SharedWriteLock, 453
samchon::library::SharedWriteLock, 455
size
samchon::WeakString, 508
samchon::library::CaseGenerator, 78
samchon::library::SQLStatement, 471
samchon::library::Semaphore, 439
size_
samchon::library::Semaphore, 439
SlaveClient, 456
SlaveServer, 458
SlaveSystem, 460
SmartPointer
samchon::SmartPointer, 463
SmartPointer< _Ty >, 462
split
samchon::WeakString, 511
samchon::library::StringUtil, 479
start
samchon::protocol::ExternalClientArray, 228

samchon::protocol::ExternalServer, 231
samchon::protocol::ExternalServerArray, 234
samchon::protocol::ExternalSystem, 237
samchon::protocol::ExternalSystemArray, 240
samchon::protocol::master::DistributedClientArray, 134
samchon::protocol::master::DistributedClientArrayMediator, 139
samchon::protocol::master::DistributedServerArray, 149
samchon::protocol::master::DistributedServerArrayMediator, 153
samchon::protocol::master::DistributedSystemArray, 169
samchon::protocol::master::DistributedSystemArrayMediator, 175
samchon::protocol::master::ParallelClientArrayMediator, 388
samchon::protocol::master::ParallelServerArrayMediator, 400
samchon::protocol::master::ParallelSystemArrayMediator, 424

startTime
samchon::protocol::InvokeHistory, 309

stdev_p
samchon::library::Math, 343

stdev_s
samchon::library::Math, 343

str
samchon::WeakString, 513

StringUtil, 472

substitute
samchon::library::StringUtil, 474

substituteSQL
samchon::library::StringUtil, 475

substr
samchon::WeakString, 510

substring
samchon::WeakString, 510
samchon::library::StringUtil, 478

SystemRole, 480

TAG
samchon::example::chat_service::ChatMessage, 87
samchon::example::chat_service::ChatRoom, 90
samchon::library::FTInstance, 261
samchon::namtree::NTCriteria, 351
samchon::namtree::NTParameter, 365
samchon::namtree::NTParameterArray, 368
samchon::namtree::NTParameterDetermined, 371
samchon::namtree::NTSide, 374
samchon::protocol::Entity, 187
samchon::protocol::ExternalSystem, 238
samchon::protocol::ExternalSystemArray, 242
samchon::protocol::ExternalSystemRole, 245
samchon::protocol::Invoke, 304
samchon::protocol::InvokeHistory, 308
samchon::protocol::InvokeHistoryArray, 311
samchon::protocol::InvokeParameter, 316
samchon::protocol::SystemRole, 481
samchon::protocol::master::DistributedSlaveSystemMediator, 160
samchon::protocol::master::ParallelSlaveSystemMediator, 410

TP_1970
samchon::library::Date, 119

TSQLStatement, 486

TSQLi, 484
samchon::library::TSQLi, 486

tag
samchon::library::XML, 526

toHTML
samchon::protocol::IHTMLEntity, 293

toInvoke
samchon::protocol::InvokeHistory, 309

toLinuxTime
samchon::library::Date, 118

toLowerCase
samchon::WeakString, 513
samchon::library::StringUtil, 479

toMatrix
samchon::library::CaseGenerator, 78

toMultibyte
samchon::library::Charset, 80, 81

toNumber
samchon::library::StringUtil, 475

toSQL
samchon::protocol::ISQLEntity, 328

toString
samchon::example::packer::Wrapper, 518
samchon::example::tsp::GeometryPoint, 281
samchon::example::tsp::Travel, 483
samchon::library::Date, 118
samchon::library::Datetime, 127
samchon::library::URLVariables, 498
samchon::library::XML, 525

toTH
samchon::protocol::IHTMLEntity, 292

toTM
samchon::library::Date, 118

toTR
samchon::protocol::IHTMLEntity, 293

toUTF8
samchon::library::Charset, 81

toUnicode
samchon::library::Charset, 82

toUpperCase
samchon::WeakString, 513
samchon::library::StringUtil, 479

toXML
samchon::example::chat_service::ChatMessage, 88
samchon::example::chat_service::ChatRoom, 91
samchon::library::FTFile, 255
samchon::library::FTFolder, 258
samchon::library::FTInstance, 262
samchon::library::SQLStatement, 471
samchon::library::TSQLStatement, 488
samchon::namtree::INTExplore, 299
samchon::namtree::NTCriteria, 351
samchon::namtree::NTFile, 360
samchon::namtree::NTParameter, 366
samchon::namtree::NTParameterDetermined, 371
samchon::namtree::NTSide, 374
samchon::protocol::Entity, 188
samchon::protocol::EntityArray, 194
samchon::protocol::EntityGroup, 206
samchon::protocol::EntityList, 212
samchon::protocol::ExternalClientArray, 228

samchon::protocol::ExternalServer, 231
samchon::protocol::ExternalSystem, 238
samchon::protocol::ExternalSystemRole, 245
samchon::protocol::Invoke, 305
samchon::protocol::InvokeHistory, 309
samchon::protocol::InvokeParameter, 316
samchon::protocol::PRInvokeHistory, 429
samchon::protocol::SystemRole, 481
samchon::protocol::master::DistributedClientArray, 134
samchon::protocol::master::DistributedClientArrayMediator, 139
samchon::protocol::master::DistributedServer, 146
samchon::protocol::master::DistributedSystem, 164
samchon::protocol::master::DistributedSystemArray, 169
samchon::protocol::master::DistributedSystemArrayMediator, 175
samchon::protocol::master::DistributedSystemRole, 178
samchon::protocol::master::ParallelClientArrayMediator, 388
samchon::protocol::master::ParallelServer, 393
samchon::protocol::master::ParallelSystem, 413
samchon::protocol::master::ParallelSystemArrayMediator, 424
Travel, 482
samchon::example::tsp::Travel, 483
trim
samchon::WeakString, 512
samchon::library::StringUtil, 476
tryAcquire
samchon::library::Semaphore, 439
tryInsert
samchon::example::packer::Wrapper, 517
samchon::example::packer::WrapperArray, 519
type
samchon::protocol::InvokeParameter, 317

URLVariables, 496
samchon::library::URLVariables, 497
UniqueAcquire, 488
samchon::library::UniqueAcquire, 490
UniqueEntityArray
samchon::protocol, 47
UniqueEntityList
samchon::protocol, 47
UniqueReadLock, 491
samchon::library::UniqueReadLock, 492, 493
UniqueWriteLock, 493
samchon::library::UniqueWriteLock, 495, 496
unlock
samchon::library::SharedReadLock, 453
samchon::library::SharedWriteLock, 456
samchon::library::UniqueReadLock, 493
samchon::library::UniqueWriteLock, 496
User, 498
samchon::protocol::service::User, 500

value
samchon::library::XML, 526
variance_p
samchon::library::Math, 344
variance_s
samchon::library::Math, 344

WeakString, 504
samchon::WeakString, 507, 508
WebServerConnector, 514
Wrapper, 515
samchon::example::packer::Wrapper, 517
WrapperArray, 518
samchon::example::packer::WrapperArray, 519
write
samchon::ByteArray, 67
writeLock
samchon::library::RWMutex, 436
writeReversely
samchon::ByteArray, 68
XML, 520
samchon::library::XML, 523, 524
XMLList
samchon::library, 30