

Priručnik radnog okruženja za pisanje protokola

Verzija 0.2

Sadržaj

1. [Uvod](#)
2. [Osnovne komponente kernela](#)
 - 2.1. [Klasa FSMSystem](#)
 - 2.1.1. [Inicijalizacija sistema automata](#)
 - 2.1.2. [Pokretanje sistema automata](#)
 - 2.2. [Klasa FiniteStateMachine](#)
3. [Rukovanje vremenskim kontrolama](#)
4. [Rukovanje memorijom](#)
5. [Rukovanje porukama](#)
6. [Podrška za rad sa TCP/IP protokolom](#)
 - 6.1. [Klase FSMSystemWithTCP](#)
 - 6.2. [Klasa NetFSM](#)
7. [Konstante, tipovi podataka i funkcije](#)
8. [API Funkcije](#)
9. [Primer realizacije dva automata](#)
10. [Primer realizacije NetFSM](#)

[Skraćenice](#)

[Literatura](#)

1. Uvod

Osnovna primena radnog okruženja je realizacija konačnih automata koji se koriste za implementaciju telekomunikacionih protokola. Realizovani automati se povezuju preko sistema automata (FSM sistem) koji rukuje resursima koji su potrebni za rad protokola. Implementacija određenog programskog rešenja svodi se na realizaciju funkcija prelaza. Funkcija prelaza se sastoji od procedura koje obrađuju primljenu poruku u datom stanju automata na osnovu kojih automat prelazi u novo stanje i generiše odgovarajuće poruke (izlaze iz datog stanja) prema drugim automatima u sistemu automata. Ove funkcije potrebno je definisati za sve automate koji će biti uključeni u sistem automata. Za ispravan rad sistema automata potrebno je pravilno inicijalizovati sve elemente kernela koji će biti korišćeni.

2. Osnovne komponente kernela

Kernel je realizovan u programskom jeziku C++ primenom obejktno orjentisanog pristupa. Osnovne komponente kernela su realizovane kao klase koje daju osnovnu funkcionalnost automatima i sistemu automata. Funkcionalnost automata koji su potrebni za konkretan problem se ostvaruje kroz nasleđivanje bazne klase [FiniteStateMachine](#). U izvedenoj klasi potrebno je: realizovati virtualne funkcije, realizovati nove funkcije koje rešavaju dati problem (funkcije prelaza) i eventualno je potrebno redefinisati neke nasledene funkcije kako bi se izmenila osnovna funkcionalnost bazne klase. Osnovne klase kernela su:

- Klasa [FSMSystem](#),
- Klasa [FiniteStateMachine](#),

2.1. Klasa FSMSystem

Instanca klase FSMSystem je objekat koji predstavlja sistem automata. Zaštićeni atributi ove klase predstavljaju resurse koji stoje na raspolaganju automatima koji su uključeni u sastav sistema automata. Osnovni zadatak ove klase je rukovanje porukama, memorijom i vremenskim kontrolama. Pravilno funkcionisanje sistema automata podrazumeva sledeći niz koraka:

- Deklaracija sistema automata,
- Inicijalizacija sistema automata,
- Pokretanje sistema automata,
- Zaustavljanje sistema automata,

2.1.1. Inicijalizacija sistema automata

Inicijalizacija sistema automata se sastoji od sledećih koraka:

- Stvaranje sistema automata, vidi konstruktor [FSMSystem\(\)](#),
- Stvaranje i inicijalizacija instanci automata, vidi konstruktor [FiniteStateMachine\(\)](#),
- Dodavanje automata u sistem automata,
- Inicijalizacija kernela,
- Evidencija rada sistema automata,

Za stvaranje sistema automata potrebno je obezbediti dva parametra: broj tipova automata i broj poštanskih sandučića u kojima će poruke čekati na obradu. Dodavanje instanci automata u sistem automata se izvršava pozivom jedne od metoda:

```

void Add( ptrFiniteStateMachine object,    // Adresa instance automata
          uint8 automateType,              // Tip automata
          uint32 numOfObjects,              // Broj instanci
          bool useFreeList = false );      // Lista slobodnih automata

void Add( ptrFiniteStateMachine object,    // Adresa instance automata
          uint8 automateType );            // Tip automata

```

Pri čemu se prva instanca automata za zvaki tip dodaje prvom funkcijom a ostali automati datog tipa se dodaju sa drugom funkcijom. Inicijalizacija kernela se izvodi pozivom metode:

```

void InitKernel(      uint8 buffClassNo,        // Broj različitih tipova
                      uint32 *buffersCount,      // Broj bafera po tipu
                      uint32 *buffersLength,      // Dužina bafera po tipu
                      uint8 numOfMboxes=0,       // Broj poštanskih sandučića
                      TimerResolutionEnum timerRes = Timer1s );
                      // Rezolucija vremeskih kontrola

```

Evidencija rada sistema automata obezbeđuje da se vodi evidencija poruka koje automati razmenjuju kao i evidencija sadržaja poruka koje se razmenjuju. Poruke se evidentiraju automatski u datoteci koja se definisana prilikom inicijalizacije. Datoteka log.ini je opcionalna i u njoj se definišu tekstualni nazivi za poruke koje automati razmenjuju. Ove tekstualne poruke će se koristiti u toku evidentiranja poruka umesto stvarnih heksadecimalnih vrednosti što značajno olakšava praćenje rada sistema automata. Datoteka log.ini se mora nalaziti u sistemskom direktorijumu (C:\WINNT ili C:\WINDOWS). Format ove datoteke je sledeći:

```

[AUTOMATES]
1=AUTOMAT1_FSM
2=AUTOMAT2_FSM
R.Br.=TIP_AUTOMATA
[MESSAGES]
0=0xe000,MSG_1,0
1=0xe002,MSG_2,0
R.Br.=KOD_PORUKE,TEXTUALNI_NAZIV,0

```

Primer:

```

...
#define BROJ_BAFERA          3
#define BROJ_AUTOMATA_1     5
#define BROJ_AUTOMATA_2     9
...

// Definicija bafera: tri tipa, pri čemu je broj bafera po tipovima
// 50, 30 i 20 i gde je njihov broj 128, 256 i 512 bajtova
uint8  buffClassNo          = BROJ_BAFERA;
uint32 buffersCount[BROJ_BAFERA] = { 50, 30, 20 };
uint32 buffersLength[BROJ_BAFERA] = { 128, 256, 512 };

// Stvaranje sistema automata koji ima dva tipa automata i koristi
// dva poštanska sandučeta, za svaki tipa automata posebno
FSMSystem *sistemAutomata = new FSMSystem( 2, 2 );

```

```

// Stvaranje instanci automata
Automat1 *automat1 = new Automat1[BROJ_AUTOMATA_1];
Automat2 *automat2 = new Automat2[BROJ_AUTOMATA_2];

// Dodavanje instanci automata u sistem automata i njihova implicitna
// inicijalizacija ovde se poziva metoda Initialize\(\) za svaku instancu
// automata.
sistemAutomata->Add(&automat1[0], AUTOMAT1_FSM, BROJ_AUTOMATA_1, false);
for( i=1; i<BROJ_AUTOMATA_1; i++ )
    sistemAutomata->Add( &automat1[i], AUTOMAT1_FSM );

sistemAutomata->Add(&automat2[0], AUTOMAT2_FSM, BROJ_AUTOMATA_2, true);
for( i=1; i<BROJ_AUTOMATA_2; i++ )
    sistemAutomata->Add( &automat2[i], AUTOMAT2_FSM );

// Inicijalizacija kernela
sistemAutomata->InitKernel(buffClassNo, buffersCount, buffersLength, 2);

// Inicijalizacija objekata za evidenciju rada sistema automata
lf = new LogFile( "log.log", "log.ini" );
LogAutomateNew::SetLogInterface( lf );
...

```

2.1.2. Pokretanje sistema automata

Sistem automata se pušta u rad metodom [Start\(\)](#). Najčešće se ova metoda poziva u posebnoj programskoj niti.

Primer:

```

...
DWORD WINAPI SistemAutomataThreadFunc( void* param ){
    try {
        sistemAutomata->Start();
    }
    catch(...){
        OutputDebugString("Exception - prekid rada sistema automata\n");
        return 0;
    }
    OutputDebugString("Kraj rada sistema automata\n");
    return 0;
}
...
// Negde u main funkciji
DWORD sistemAutomataThreadId;
CreateThread(NULL, 0, SistemAutomataThreadFunc, 0, 0, sistemAutomataThreadId);
...

```

2.2. Klasa FiniteStateMachine

Svi automati koji treba da se uključe u sistem automata se realizuju tako što se nasledi bazna klasa FiniteStateMachine. U okviru bazne klase definisan je skup virtuelnih metoda koje je potrebno implementirati kako bi automat pravilno funkcionisao, te metode su:

```

1. MessageInterface *GetMessageInterface( uint32 id );
2. void SetDefaultHeader( uint8 infoCoding );
3. uint8 GetMbxId();
4. uint8 GetAutomate();
5. void SetDefaultFSMData();
6. void NoFreeInstances();
7. void Initialize();

```

Definicija ovih metoda koje se koriste u najvećem broju slučajeva prikazana je kroz primer. Opširnije o ovim funkcijama može se naći u poglavlju gde je dat opis svih funkcija.

Primer:

```

...
// Ova metoda sadrzi informacije o tome koje poruke automat može da obradi
// Za ovakvu definiciju u okviru klase potrebno je definisati dodatnu
// promenljivu: StandardMessage standardMsgCoding;
MessageInterface *Automat::GetMessageInterface( uint32 id ){
    switch(id){
        case 0x00:
            return &StandardMsgCoding;

            // Ostale definicije
            // case 0x01:
            // case 0x02:
    }
    throw TErrorObject( __LINE__, __FILE__, 0x01010400 );
}

// Ova metoda popunjava zaglavlje poruke koju automati medjusobno razmenjuju
void Automat::SetDefaultHeader( uint8 infoCoding ){
    SetMsgInfoCoding( infoCoding );
    SetMessageFromData();
}

// Metoda kojom se definiše broj poštanskog sandučeta iz kog će automati ove
// klase primiti poruke koje su pristigle u sistem automata.
uint8 Automat::GetMbxId(){
    return AUTOMATE_MBX_ID;
}

// Metoda vraća broj koji identifikuje tip automata kojem pripada ovaj automat
uint8 Automat::GetAutomate(){
    return AUTOMATE_TYPE_ID;
}

// Metoda u kojoj se izvršava inicijalizacija atributa instanci ove klase.
void Automat::SetDefaultFSMData(){
    atribut1 = VREDNOST_1;
    atribut2 = VREDNOST_2;
}

// Metoda koja će se pozvati ako nema više slobodnih automata da obrade
// pristiglu poruku. Ova metoda se koristi ako su automati ove klase prethodno
// dodati u sistem automata sa parametrom useFreeList = true
void Automat::NoFreeInstances(){
    // Aktivnost ukoliko nema slobodnih automata u sistemu
}

```

```
// Metoda koja inicijalizuje funkcije prelaza i definiše vremenske kontrole
// koje će automati ove klase koristiti u svom radu. Ova metoda se poziva
// implicitno u okviru procesa dodavanja instanci automata u sistem automata.
// Sve funkcije prelaza se posebno deklarise i definise.
void Automat::Initialize(){
    // Ovde ide serija inicijalizacija:
    // InitEventProc(uint8 state, uint16 event, PROC_FUN_PTR fun);
    // InitUnexpectedEventProc(uint8 state, PROC_FUN_PTR fun);
    // InitTimerBlock(uint16 timerId, uint32 timerCount, uint16 signalId);

    InitEventProc(IDLE, MSG_SEND, (PROC_FUN_PTR) &Automat::Idle_MsgSend);
    InitEventProc(IDLE, MSG_RCV, (PROC_FUN_PTR) &Automat::Idle_MsgReceive);

    InitEventProc(SEND, MSG_NEW, (PROC_FUN_PTR) &Automat::Send_MsgNew);
    InitEventProc(SEND, MSG_END, (PROC_FUN_PTR) &Automat::Send_MsgEnd);
    InitEventProc(IDLE, T200_CODE, (PROC_FUN_PTR) &Automat::T200Expired);

    InitUnexpectedEventProc(IDLE, (PROC_FUN_PTR) &Automat::Idle_Unexpected);
    InitUnexpectedEventProc(SEND, (PROC_FUN_PTR) &Automat::Send_Unexpected);

    InitTimerBlock( T200, T200_VALUE, T200_CODE );
}
```

Atributi i stanja automata date klase se definišu u zavisnosti od problema koji se rešava. Funkcije prelaza se pozivaju kada automat primi odgovarajuću poruku dok se nalazi u određenom stanju što je definisano metodom [Initialize\(\)](#). Svaka funkcija prelaza se posebno definiše kao metoda klase u okviru koje se izvršava odgovarajuća obrada primljene poruke u datom stanju.

Vremenske kontrole koje će automat koristiti u svom radu deklarise se u metodi [Initialize\(\)](#). Ovom deklaracijom se u kernelu inicijalizuju odgovarajuće strukture podataka koje sadrže osnovne podatke o vremenskoj kontroli koju će automat koristiti. To su: naziv vremenske kontrole, vreme u sekundama do isteka vremenske kontrole i naziv (kod) poruke koja će biti automatski poslata automatu kada ova vremenska kontrola istekne.

3. Rukovanje vremenskim kontrolama

Tokom inicijalizacije sistema automata inicijalizuje se i rukovanje vremenskim kontrolama. Automati koji koriste vremenske kontrole u svom radu pozivaju API funkcije kernela pomoću kojih se inicijalizuju odgovarajuće strukture podataka. Funkcije za rukovanje vremenskim kontrolama se sledeće:

```
1. void InitTimerBlock( uint16 tmrId, uint32 count, uint16 signalId );
2. void StartTimer( uint16 tmrId );
3. void StopTimer( uint16 tmrId );
4. void RestartTimer( uint16 tmrId )
5. bool IsTimerRunning( uint16 tmeId );
```

Funkcija [InitTimerBlock\(\)](#) kojom se izvršava inicijalizaciju vremenske kontrole prikazana je u klasi [FiniteStateMachine](#) u okviru metode [Initialize\(\)](#). Sve funkcije se izvršavaju za vremensku kontrolu sa identifikacionim brojem tmrId koji se prosleđuje svakoj funkciji posebno kao parametar. Funkcija [StartTimer\(\)](#) pokreće vremensku kontrolu, [StopTimer\(\)](#) zaustavlja vremensku kontrolu, [RestartTimer\(\)](#) prvo zaustavlja vremensku kontrolu i zatim je ponovo pokreće i funkcija [IsTimerRunning\(\)](#) se koristi ako je potrebno proveriti da li je neka vremenska kontrola pokrenuta.

Primer:

```
...
if( !IsTimerRunning( T200 ) )
{
    StartTimer( T200 );
}
else
    StopTimer( T200 );
...
```

Rukovanje vremenskim kontrolama u okviru sistema automata funkcioniše tako što automati koji koriste vremenske kontrole u svom radu koriste navedene funkcije da aktiviraju odgovarajuće vremenske kontrole sistema automata. Vremenske kontrole sistema automata po isteku definisanog vremenskog perioda interno generišu odgovarajuće poruke prema automatu koji je datu vremensku kontrolu pokrenuo kako bi on obradio taj događaj preko svojih funkcija prelaza.

4. Rukovanje memorijom

Pošto kernel nalazi svoju osnovnu primenu u sistemima koji rade u realnom vremenu, zauzimanje i oslobađanje memorije se ne može prepustiti operativnom sistemu. Zbog toga je rukovanje memorijom jedna od osnovnih funkcija kernela.

Rukovanje memorijom je ostvareno preko bafera. Veličina bafera i njihov broj su definisani prilikom inicijalizacije sistema automata. Baferi koji se nalaze u satavu kernela se koriste indirektno prilikom slanja poruka i za rad vremenskih kontrola ali se mogu koristiti i direktno i to uz pomoć sledećih funkcija:

```
1. uint8 *GetBuffer( uint32 length );
2. void RetBuffer( uint8 *buff );
3. bool IsBufferSmall( uint8 *buff, uint32 length );
4. uint32 GetBufferLength( uint8 *buff );
```

Bafer se preuzima od kernela pozivom funkcije [GetBuffer\(\)](#) sa parametrom koji predstavlja minimalnu veličinu traženog bafera. Svi baferi koji su uzeti od kernela moraju se vratiti funkcijom [RetBuffer\(\)](#), u suprotnom će se gubiti baferi i rad kernela neće biti ispravan. Pored ove dve funkcije postoje i dve funkcije koje daju informacije o baferu koji je već preuzet od kernela. [IsBufferSmall\(\)](#) proverava da li je dati bafer veći od zadate veličine dok funkcija [GetBufferLength\(\)](#) vraća tačnu dužinu datog bafera.

Primer:

```
...
// Definišemo grupe bafera, prva grupa sa 10 bafera veličine 128 bajtova i
// druga grupa sa 15 bafera veličine 256 bajtova
uint8 buffClassNo      = 2;
uint32 buffersCount[2] = { 10, 15 };
uint32 buffersLength[2] = { 128, 256 };
...

// Inicijalizacija kernela (brojMBX je nebitan za ovaj primer)
sistemAutomata->InitKernel( buffClassNo, buffersCount, buffersLength, brojMBX );
...

uint32 bufferLength;
uint8 *pokazivac = GetBuffer( 100 );
if( IsBufferSmall( pokazivac, 129 ) )
{
    RetBuffer( pokazivac );
    pokazivac = GetBuffer(129);
}
if( pokazivac != NULL )
    bufferLength = GetBufferLength( pokazivac );
...
```

5. Rukovanje porukama

Osnovna komunikacija između automata koji su uključeni u sistem automata se ostvaruje preko poruka koje automati međusobno razmenjuju. Poruke koje su poslate u okviru sistema automata se smeštaju u odgovarajuće poštanske sandučice, i tu poruke čekaju trenutak kada će biti obrađene.

Obrada primljenih poruka se odvija automatski od strane kernela u okviru sistema automata, i svodi se na to da se poruke uzimaju redom jedna po jedna iz svakog poštanskog sandučeta. Nakon što se iz poruke ustanovi tačna instanca automata kojem je upućena poruka, na osnovu koda poruke, poziva se odgovarajuća funkcija obrade.

API funkcije kernela za rukovanje porukama se mogu grupisati u dve grupe i to funkcije za rad sa primljenom porukom i funkcije za rad sa novom porukom koja tek treba da se pripremi i da se pošalje nekom automatu.

Funkcijama koje rade nad primljenom porukom se dobijaju informacije o automatu koji je poslao poruku iz zaglavlja poruke kao i vrednosti parametara koji su bili smesteni u poruci dok se funkcijama koje rade sa novom porukom izvodi priprema za slanje poruke (zauzimanje bafera, popunjavanje zaglavlja podacima o automatu koji šalje poruku), dodavaju se parametri u poruku i prebacuju poruku u odgovarajući poštanski sandučić automata kojem se šalje poruka.

Poruke se mogu poslati samo iz automata ili iz sistema automata.

Primer 1:

```
...
// Preuzimanje vrednosti parametra PARAM_1 dužine 2 bajta iz primljene poruke
WORD word; // Promenljiva koja će primiti novu vrednost
GetParamWord( PARAM_1, word );

// Preuzimanje vrednosti parametra proizvoljne dužine
// Za StandardMessage najviše 256 bajtova, ako je potrebno da parametri budu
// veće dužine potrebno je napraviti novu klasu nasleđivanjem klase
// StandardMessage i redefinisati odgovarajuće metode za rad sa porukama.
uint8 *pokazivac;
uint8 text[200];
uint8 msgLength;

pokazivac = GetParam( TEXT );
if( pokazivac != NULL )
{
    // Važi samo sa StandardMessage
    // 1 i 2 bajt su naziv parametra, 3 bajt dužina parametra u bajtovima
    // 4 bajt i dalje informacioni sadržaj parametra
    memcpy( text, pokazivac+3, *( pokazivac+2 ) );

    // Pravimo string, stavljamo nulu na kraj niza karaktera
    memset( text+*( pokazivac+2 ), 0x00, 1 );
}
```

Primer 2:

```
...
// Priprema poruke parametri: velicina bafera i tip poruke
PrepareNewMessage( 0xAA, NAZIV_PORUKE );

// Popunjavanje zaglavlja poruke:
// tip automata kome se salje, njegov ID i ako postoji grupa kojoj pripada
SetMsgToAutomate( TIP_AUTOMATA );
SetMsgObjectNumberTo( automateId );
SetMsgToGroup( INVALID_08 );

// Dodavanje parametara: vidi i druge AddParam funkcije
AddParamByte( PARAM_1, byte );
AddParamWord( PARAM_2, word );
AddParam( PARAM_3, duzinaParametra, pokazivacNaParam );

// Slanje pruke u poštansko sanduče
SendMessage( AUTOMATE_MBX_ID );
```

Primer 3:

```
...
// Ovako se šalje poruka iz sistema automata
uint8 *msg = GetBuffer( messageInfoLength+MSG_HEADER_LENGTH );

// infoBaffer mora biti pravilno formatiran kako bi se primljena
// poruka pravilno obradila
memcpy( msg+MSG_HEADER_LENGTH, infoBaffer, infoBafferLength );

SetMsgFromAutomate( AUTOMATE_TYPE_FROM_ID, msg );
SetMsgFromGroup( INVALID_08, msg );
SetMsgObjectNumberFrom( automateFromId, msg );

SetMsgToAutomate( AUTOMATE_TYPE_TO_ID, msg );
SetMsgToGroup( INVALID_08, msg );
SetMsgObjectNumberTo( automateToId, msg );

SetMsgInfoCoding( 0, msg ); // 0 = StandardMessage
SetMsgCode( MSG_FROM_SYSTEM_AUTOMATA, msg );
SetMsgInfoLength( infoBafferLength, msg );
SendMessage( AUTOMATE_TO_MBX_ID, msg );
...
```

6. Podrška za rad sa TCP/IP protokolom

Sistem automata u svojoj osnovnoj verziji podrazumeva da su svi automati instancirani na jednom računaru u okviru istog programa. U okviru programske podrške za realizaciju konačnih determinističkih automata moguće je na dva načina ostvariti prostornu distribuiranost automata. Prvi korišćenjem mehanizma distribucije, a drugi korišćenjem mrežne sprege. Prvi je namenjen prevashodno rešenjima kod kojih je programska podrška za realizaciju konačnih determinističkih automata zastupljena u svakom delu, dok u slučaju potrebe za interakcijom sa sistemima koji nisu zasnovani na navedenoj platformi, a imaju definisanu mrežnu spregu (TCP/IP) koristi se drugi pristup. Kod drugog pristupa implementirane su dve dodatne klase kojima se ostvaruje navedena interakcija.

Klase [FSMSystemWithTCP](#) i [NetFSM](#) su klase koje proširuju osnovno programsko jezgro funkcijama koje omogućuju da automati razmenjuju poruke preko TCP/IP protokola. Promene u implementaciji sistema automata podrazumevaju instanciranje klase [FSMSystemWithTCP](#) umesto klase [FSMSystem](#) dok implementacija automata podrazumeva nasleđivanje klase [NetFSM](#) umesto klase [FiniteStateMachine](#).

6.1. Klasa FSMSystemWithTCP

Ova klasa je nastala nasleđivanjem klase [FSMSystem](#) i njenim proširivanjem sa podrškom za razmenu poruka preko TCP/IP protokola. Nova klasa je zadržala sve stare funkcije i rad tog dela ove klase je identičan sa radom stare klase.

Da bi automati mogli da razmenjuju poruke preko TCP/IP protokola sistem automata sa podrškom za TCP/IP mora pre početka svog rada da uspostavi vezu sa drugim sistemom automata. Veza između dva sistema automata se uspostavlja tako što jedna ili obe strane instanciraju server funkcijom [InitTCPServer\(\)](#) koji očekuje da se ostvari veza preko TCP/IP protokola. Veza nastaje kada druga strana uputi serveru komandu za uspostavu veze pozivom funkcije [establishConnection\(\)](#).

Primer:

```
// U prvom sistemu automata
//
// Inicijalizacija kernela
sistemAutomata->InitKernel(buffClassNo,buffersCount,buffersLength, 2);

// Inicijalizacija TCP/IP servera na portu 5000
// gde je NetFSM_Automat automat koji je instanca
// klase koja nasleđuje klasu NetFSM
sistemAutomata->InitTCPServer( 5000, NetFSM_Automat );

// U drugom sistemu automata
//
// Postavljamo TCP/IP parametre servera prvog sistema automata
automatIzDrugogSistema.setPort( 5000 );
automatIzDrugogSistema.setIP( "192.168.77.77" );
automatIzDrugogSistema.establishConnection();
...
```

2. Klasa NetFSM

Ova klasa je nastala nasleđivanjem klase [FiniteStateMachine](#) i njenim proširivanjem sa podrškom za rukovanje porukama preko TCP/IP protokola. Nova klasa je zadržala sve stare funkcije dok su za rukovanje porukama preko TCP/IP protokola dodate tri nove funkcije. Nove funkcije su:

```
1. virtual uint16 convertNetToFSMMessage()=0;
2. virtual void convertFSMToNetMessage()=0;
3. virtual uint8 getProtocolInfoCoding()=0;
```

Ove funkcije se koriste za transformaciju poruka koje stižu sa mreže u poruke koje se razmenjuju unutar sistema automata kao i za transformaciju poruka iz sistema automata u poruke mreže koje se šalju drugom sistemu sa kojim je sistem automata u komunikaciji. Navedene funkcije su prave virtualne funkcije koje se moraju implementirati u klasi koja će naslediti klasu NetFSM. Takođe prilikom implementacije datih funkcija potrebno je razlikovati memorijske lokacije u kojima okruženje izdaje i očekuje poruke pre i posle transformacije respektivno. Zapravo u slučaju slanja poruke, tj. po prijemu interne poruke iz datog sistema automata implicitno se od strane okruženja poziva metoda *convertFSMToNetMessage* u okviru koje je implementator dužan da na osnovu dobijene poruke formira poruku koju će slati na mrežu. Ulazni parametri su pristigla poruka (pokazivač na pristiglu poruku) *fsmMessageS* i dužina pristigle poruke *fsmMessageSLength*. Kao izlaz se očekuje da poruka koja se šalje na mrežu bude na memorijskoj lokaciji na koju pokazuje pokazivač *protocolMessageS* i dužina poruke koja se šalje bude postavljena u promenljivu *sendMsgLength*. U slučaju pristigle poruke sa mreže koristi se metoda *convertNetToFSM* u okviru koje se očekuje od implementatora da od pristigle poruke formira poruku razumljivu sistemu automata. Ulazni parametri su pokazivač na memorijsku lokaciju gde se nalazi pristigla poruka *protocolMessageR* i dužina pristigle poruke *receivedMessageLength*. Kao izlaz se očekuje da poruka koja se šalje u sistem automata bude na memorijskoj lokaciji na koju pokazuje pokazivač *fsmMessageR* i da se dužina te poruke postavi u promenljivu *fsmMessageRLength*.

Prethodno navedenim definišu se potrebne radnje pretvaranja poruka jednog formata u drugi čime je ostvarena neograničena pristupnost datom sistemu automata od strane drugih sistema koji imaju definisanu mrežnu (TCP) spregu. Slanje poruke na mrežu vrši se pozivom funkcija [sendToTCP](#)(). Bitno je reći da ova metoda izbacije izuzetak ako je došlo do greške, jedan od primera je slanje poruke preko veze koju je druga strana u datom trenutku prekinula.

Primer:

```
...
// Priprema poruke parametri: velicina bafera i tip poruke
PrepareNewMessage( 0xAA, NAZIV_PORUKE );

// Popunjavanje zaglavlja poruke:
// tip automata kome se salje, njegov ID i ako postoji grupa kojoj pripada
SetMsgToAutomate( TIP_AUTOMATA );
SetMsgObjectNumberTo( automateId );
SetMsgToGroup( INVALID_08 );

// Dodavanje parametara: vidi i druge AddParam funkcije
AddParamByte( PARAM_1, byte );
AddParamWord( PARAM_2, word );
AddParam( PARAM_3, duzinaParametra, pokazivacNaParam );

// Slanje pruke u poštansko sanduče
// SendMessage( AUTOMATE_MBX_ID );
// ili slanje pruke u drugi sistem automata
sendToTCP();
```

7. Konstante, tipovi podataka i funkcije

U datoteci kernelConsts.h definisane su konstante koje koristi kernel. Osnovne konstante i njihove vrednosti su:

```
- MSG_FROM_AUTOMATE    = 0;           // Kod tipa automata koji šalje poruku (BYTE)
- MSG_FROM_GROUP       = 1;           // Kod grupe automata koji šalje poruku (BYTE)
- MSG_TO_AUTOMATE      = 2;           // Kod tipa automata koji prima poruku (BYTE)
- MSG_TO_GROUP         = 3;           // Kod grupe automata koji prima poruku (BYTE)
- MSG_CODE              = 4;           // Kod poruke (WORD)
- MSG_OBJECT_ID_FROM   = 6;           // ID automata koji šalje poruku (DWORD)
- MSG_OBJECT_ID_TO     = 10;          // ID automata koji prima poruku (DWORD)
- CALL_ID              = 14;          // Jedinstveni ID poziva
- MSG_INFO_CODING      = 18;          // Tip kodiranja poruke, 0 = StandardMessage
- MSG_LENGTH           = 19;          // Dužina informacionog dela poruke
- MSG_INFO             = 21;          // Početak informacionog dela poruke
- MSG_HEADER_END       = MSG_INFO;    // Kraj zaglavlja poruke

- INVALID_08 = 0xff;                  // Za 8 bita
- INVALID_16 = 0xffff;                // Za 16 bita
- INVALID_32 = 0xffffffff;            // Za 32 bita
```

Osnovni tipovi podataka koji se koriste u kernelu su:

```
- int8,
- uint8,           // BYTE
- int16,
- uint16,          // WORD
- int32,
- uint32,          // DWORD
```

Pomoćne funkcije za konverziju tipova:

```
- void SetUInt16(uint8 *addr, uint16 value);
- void SetUInt32(uint8 *addr, uint32 value);
- uint16 GetUInt16(uint8 *addr);
- uint32 GetUInt32(uint8 *addr);
```

8. API Funkcije

Funkcije radnog okruženja za pisanje protokola su grupisane na sledeći način:

1. [Konstruktor klase FSMSystem](#)
2. [Metode klase FSMSystem](#)
3. [Konstruktor klase FSMSystemWithTCP](#)
4. [Metode klase FSMSystemWithTCP](#)
5. [Konstruktor klase FiniteStateMachine](#)
6. [Metode klase FiniteStateMachine](#)
7. [Konstruktor klase NetFSM](#)
8. [Metode klase NetFSM](#)

| Konstruktor klase FSMSystem | |
|---|--|
| FSMSystem (uint8 numOfAutomates, uint8 numberOfMbx) | Inicijalizuje objekat koji predstavlja sistem automata kao i strukture podataka koje su potrebne za njegov pravilan rad. |

| Metode klase FSMSystem | |
|------------------------|---|
| void | Add (ptrFiniteStateMachine object, uint8 automateType, uint32 numOfObjects, bool useFreeList = false) Prvi automat svakog tipa se dodaje u sistem automata ovom funkcijom. |
| void | Add (ptrFiniteStateMachine object, uint8 automateType) Svi automati, osim prvog od svakog tipa, se dodaju u sistem automata ovom funkcijom. |
| void | InitKernel (uint8 buffClassNo, uint32 *buffersCount, uint32 *buffersLength, uint8 numOfMbxs=0, TimerResolutionEnum timerRes=Timer1s) Inicijalizuje elemente kernela za rukovanje porukama, memorijom, i vremenskim kontrolama. |
| void | Remove (uint8 automateType) Funkcija uklanja sve automate datog tipa iz sistema automata. |
| ptrFiniteStateMachine | Remove (uint8 automateType, uint32 object) Funkcija uklanja automat datog tipa koji ima odgovarajući identifikacioni broj iz sistema automata. |
| virtual void | Start () Ova funkcija pokreće rad sistema automata. |
| void | StopSystem () Ova funkcija zaustavlja rad sistema automata. |

| Konstruktor klase FSMSystemWithTCP | |
|--|---|
| FSMSystemWithTCP (uint8 numOfAutomates, uint8 numberOfMbx) | Inicijalizuje objekat koji predstavlja sistem automata koji ima podršku za rad sa TCP/IP protokolom kao i strukture podataka koje su potrebne za njegov pravilan rad. |

| Metode klase FSMSystemWithTCP | |
|-------------------------------|--|
| int | InitTCPServer (uint16 port, uint8 automateType, char *ipAddress = 0, unsigned char *parm = 0, int length = 0) Funkcija inicijalizuje server koji čeka da stigne poziv od drugog sistema automata kako bi se uspostavila veza dva sistema automata preko TCP/IP protokola. |

Konstruktor klase FiniteStateMachine

[FiniteStateMachine](#)(uint16 numOfTimers = DEFAULT_TIMER_NO, uint16 numOfState = DEFAULT_STATE_NO, uint16 maxNumOfProceduresPerState = DEFAULT_PROCEDURE_NO_PER_STATE, bool getMemory = true)

Inicijalizuje instancu datog automata i strukture podataka koje su potrebne za njegov pravilan rad.

Metode klase FiniteStateMachine

| | |
|--------------------------|--|
| uint8* | <u>AddParam</u> (uint16 paramCode, uint32 paramLength, uint8 *param) Funkcija AddParam() se koristi za dodavanje parametra u novu poruku. |
| uint8* | <u>AddParamByte</u> (uint16 paramCode, BYTE param) Funkcija se koristi za dodavanje parametra veličine jednog bajta u novu poruku. |
| uint8* | <u>AddParamDWord</u> (uint16 paramCode, DWORD param) Funkcija se koristi za dodavanje parametra veličine četiri bajta u novu poruku. |
| uint8* | <u>AddParamWord</u> (uint16 paramCode, WORD param) Funkcija se koristi za dodavanje parametra veličine dva bajta u novu poruku. |
| virtual void | <u>CheckBufferSize</u> (uint32 paramLength) Obezbeđuje bafer dovoljno velik da u njega može da se smesti dati parametar. |
| virtual void | <u>ClearMessage</u> () Funkcija vraća nazad programskom jezgri bafer primljene poruke i dodeljuje vrednost NULL pokazivaču na primljenu poruku. |
| virtual void | <u>CopyMessage</u> () Funkcija se koristi da napravi kopiju primljene poruke. |
| virtual void | <u>CopyMessage</u> (uint8 *msg) Pravi kopiju poruke koja je smeštena u baferu čija je adresa data parametrom. |
| virtual void | <u>CopyMessageInfo</u> (uint8 infoCoding, uint16 lengthCorrection = 0) Funkcija se koristi da kopira informacioni deo primljene poruke (bez zaglavlja). |
| virtual void | <u>Discard</u> (uint8* buff) Briše poruku koja je smeštena u datom baferu i bafer se vraća nazad kernelu. |
| void | <u>DoNothing</u> () Ova funkcija se poziva kada u automat stigne ne očekivana poruka. |
| void | <u>FreeFSM</u> () Javlja sistemu automata da je automat završio svoju obradu i da je automat slobodan. |
| virtual uint8 | <u>GetAutomate</u> ()=0 Funkcija GetAutomate() vraća vrednost koja identifikuje tip automata. |
| uint8 | <u>GetBitParamByteBasic</u> (uint32 offset, uint32 mask=MASK_32_BIT) Vraća maskiranu vrednost jednog bajta koja se nalazi u primljenoj poruci. |
| uint32 | <u>GetBitParamDWordBasic</u> (uint32 offset, uint32 mask=MASK_32_BIT) Vraća maskiranu vrednost četiri bajta koja se nalaze u primljenoj poruci. |
| uint16 | <u>GetBitParamWordBasic</u> (uint32 offset, uint32 mask=MASK_32_BIT) Vraća maskiranu vrednost dva bajta koja se nalaze u primljenoj poruci. |
| virtual uint8* | <u>GetBuffer</u> (uint32 length) Vraća adresu bafera koji je veći ili jednak od vrednosti zadate parametrom length. |
| uint32 | <u>GetBufferLength</u> (uint8 *buff) Funkcija GetBufferLength() vraća veličinu bafera u bajtovima. |
| virtual inline uint32 | <u>GetCallId</u> () Vraća vrednost koja se koristi za praćenje toka poziva u kojem automat trenutno učestvuje. |
| uint32 | <u>GetCount</u> (uint8 mbx) Funkcija vraća broj poruka u poštanskom sandučetu sa koji je dat parametrom mbx. |
| virtual uint8 | <u>GetGroup</u> () Funkcija GetGroup() vraća broj koji identifikuje grupu kojoj ovaj automat pripada. |
| virtual uint8 | <u>GetInitialState</u> () Funkcija vraća broj koji identifikuje inicijalno stanje u kojem se automat nalazi. |
| virtual inline uint8 | <u>GetLeftMbx</u> () Vraća broj poštanskog sandučeta automata koji je levo od datog automata. |

| | |
|------------------------------|---|
| virtual inline uint8 | GetLeftAutomate() Funkcija vraća broj tipa automata koji se nalazi levo od datog automata. |
| virtual inline uint8 | GetLeftGroup() Vraća broj grupe kojoj pripadaju automati koji se nalaze levo od datog automata. |
| virtual inline uint32 | GetLeftObjectId() Vraća broj koji identifikuje automat koji se nalazi levo od datog automata. |
| virtual uint8 | GetMbxId() Funkcija vraća broj poštanskog sandučeta u koji stižu poruke za dati automat. |
| virtual MessageInterface* | GetMessageInterface(uint32 id) Preuzima se od automata objekat koji implementira klasu MessageInterface, a kojim se definiše kako se kodiraju poruke za dati automat u okviru sistema automata. |
| uint8* | GetMsg() Uzima prvu poruku iz poštanskog sandučeta za prijem poruka ovog automata. |
| static uint8* | GetMsg(uint8 mbx) Uzima prvu poruku iz poštanskog sandučeta koji je određen parametrom <i>mbx</i> . |
| inline uint32 | GetMsgCallId() Vraća jedinstveni identifikacioni broj poziva. |
| inline uint16 | GetMsgCode() Funkcija vraća kod poruke koji se preuzima iz zaglavlja primljene poruke. |
| inline uint8 | GetMsgFromAutomate() Vraća tip automata koji je poslao poruku. |
| inline uint8 | GetMsgFromGroup() Vraća broj grupe kojoj pripada automat koji je poslao poruku. |
| inline uint8 | GetMsgInfoCoding() Funkcija vraća broj koji identifikuje tip kodiranja poruke. |
| inline uint16 | GetMsgInfoLength() Funkcija GetMsgInfoLength() vraća dužinu informacionog dela primljene poruke. |
| inline uint16 | GetMsgInfoLength(uint8 *msg) Vraća dužinu informacionog dela poruke koja se nalazi u baferu čija je adresa data parametrom <i>msg</i> . |
| inline uint32 | GetMsgObjectNumberFrom() Funkcija vraća identifikacioni broj automata koji je poslao poruku. |
| inline uint32 | GetMsgObjectNumberTo() Funkcija vraća identifikacioni broj automata kojem je poslata poruka. |
| inline uint8 | GetMsgToAutomate() Funkcija vraća identifikacioni broj tipa automata kojem je poslata poruka. |
| inline uint8 | GetMsgToGroup() Vraća identifikacioni broj grupe kojoj automat kojem je poslata poruka pripada. |
| inline uint8* | GetNewMessage() Funkcija GetNewMessage() vraća adresu bafera u kojem se nalazi nova poruka. |
| inline uint16 | GetNewMsgInfoLength() Funkcija GetNewMsgInfoLength() vraća dužinu informacionog dela nove poruke. |
| inline uint8 | GetNewMsgInfoCoding() Funkcija vraća broj koji identifikuje vrstu kodiranja nove poruke. |
| uint8* | GetNextParam(uint16 paramCode) Vraća adresu sledeće instance traženog parametara unutar primljene poruke. |
| bool | GetNextParamByte(uint16 paramCode, BYTE &param) Vraća vrednost TRUE ako je pronađena sledeća instanca parametra zadata vrednošću <i>paramCode</i> i njegova vrednost dodeljena promenljivoj <i>param</i> . |
| bool | GetNextParamDWord(uint16 paramCode, DWORD &param) Vraća vrednost TRUE ako je pronađena sledeća instanca parametra zadata vrednošću <i>paramCode</i> i njegova vrednost dodeljena promenljivoj <i>param</i> . |
| bool | GetNextParamWord(uint16 paramCode, WORD &param) Vraća vrednost TRUE ako je pronađena sledeća instanca parametra zadata vrednošću <i>paramCode</i> i njegova vrednost dodeljena promenljivoj <i>param</i> . |

| | |
|--------------------------|--|
| virtual uint32 | GetObjectId() Funkcija vraća vrednost jedinstvenog identifikacionog broja za dati tip automata. |
| uint8* | GetParam (uint16 paramCode) Vraća adresu traženog parametra u primljenoj poruci. |
| bool | GetParamByte (uint16 paramCode, BYTE ¶m) Vraća vrednost TRUE ako je pronađen parametar zadat vrednošću <i>paramCode</i> i njegova vrednost dodeljena promenljivoj <i>param</i> . |
| bool | GetParamDWord (uint16 paramCode, DWORD ¶m) Funkcija GetParamDWord() vraća vrednost TRUE ako je pronađen parametar zadat vrednošću <i>paramCode</i> i njegova vrednost dodeljena promenljivoj <i>param</i> . |
| bool | GetParamWord (uint16 paramCode, WORD ¶m) Funkcija GetParamWord() vraća vrednost TRUE ako je pronađen parametar zadat vrednošću <i>paramCode</i> i njegova vrednost dodeljena promenljivoj <i>param</i> . |
| PROC_FUN_PTR | GetProcedure (uint16 event) Vraća adresu funkcije koja je definisana da obradi događaj dat parametrom <i>event</i> u datom stanju automata. |
| virtual inline uint8 | GetRightMbx () Vraća broj poštanskog sandučeta desnog automata. |
| virtual inline uint8 | GetRightAutomate () Vraća broj koji identifikuje tip desnog automata. |
| virtual inline uint8 | GetRightGroup () Vraća broj koji identifikuje grupu kojoj pripadaju desni automati. |
| virtual inline uint32 | GetRightObjectId (); Vraća broj koji identifikuje desni automat. |
| virtual inline uint8 | GetState () Funkcija GetState() vraća broj koji identifikuje trenutno stanje automata. |
| virtual bool | IsBufferSmall (uint8 *buff, uint32 length) Vraća vrednost TRUE ako je vrednost parametra <i>length</i> veća nego što je veličina bafera čija je adresa smeštena u parametru <i>buff</i> . |
| virtual void | Initialize () U okviru ove funkcije definišu se funkcije za obradu primljenih poruka i vremenske kontrole koje koriste automati date klase. |
| void | InitEventProc (uint8 state, uint16 event, PROC_FUN_PTR fun) Funkcijom se definiše koja funkcija će biti pozvana da obradi primljenu poruku u zavisnosti od stanja i koda primljene poruke. |
| void | InitTimerBlock (uint16 tmrId, uint32 count, uint16 signalId) Inicijalizuje elemente kernela za rukovanje vremenskom kontrolom. |
| void | InitUnexpectedEventProc (uint8 state, PROC_FUN_PTR fun) Definiše funkciju za obradu ne očekivanih poruka u odgovarajućem stanju. |
| bool | IsTimerRunning (uint16 id) Funkcija se koristi za testiranje da li je data vremenska kontrola pokrenuta. |
| void | NoFreeObjectProcedure (uint8 *msg) Definiše ponašanje automata u slučaju kada se koristi lista slobodnih automata i kada je ova lista prazna (svi automati su zauzeti). |
| virtual void | NoFreeInstances () Definiše ponašanje sistema automata u slučaju kada se koristi lista slobodnih automata i kada je ova lista prazna (svi automati su zauzeti). |
| virtual bool | ParseMessage (uint8 *msg) Proverava da li je primljena poruka kodirana na odgovarajući način i dodeljuje adresu bafera u kojem se nalazi primljena poruka promenljivoj <i>CurrentMessage</i> . |
| virtual void | PrepareNewMessage (uint8 *msg) Funkcija dodeljuje adresu bafera datog parametrom <i>msg</i> , internoj promenljivoj <i>NewMessage</i> koja će se koristiti prilikom formiranja nove poruke. |
| virtual void | PrepareNewMessage (uint32 length, uint16 code, uint8 infoCode = |

| | |
|--------------|--|
| | LOCAL_PARAM_CODING) Funkcija se koristi pri formiranju nove poruke. |
| virtual void | Process (uint8 *msg) Funkcija izvršava pripremu za obradu primljene poruke i odabira funkciju za obradu primljenu poruke u zavisnosti od koda poruke i stanja automata. |
| void | PurgeMailBox () Izbacuje sve poruke iz poštanskog sandučeta za dati tip automata. |
| bool | RemoveParam (uint16 paramCode) Funkcija uklanja parametar iz nove poruke. |
| virtual void | Reset () Funkcija dovodi automat u njegovo inicijalno stanje. |
| void | ResetTimer (uint16 id) Funkcija u internoj strukturi podataka sistema automata označava da je vraćen bafer koji je ta vremenska kontrola uzela kada je startovana. |
| void | RestartTimer (uint16 tmrId) Se koristi kad je potrebno da se vremenska kontrola zaustavi i ponovo pokrene. |
| virtual void | RetBuffer (uint8 *buff) Funkcija se koristi za vraćanje bafera koji su preuzeti od sistema automata. |
| void | ReturnMsg (uint8 mbxId) Generise novu poruku koja predstavlja kopiju primljene poruke i šalje je u poštansko sanduče koje je definisano parametrom funkcije. |
| void | SetBitParamByteBasic (BYTE param, uint32 offset, uint32 mask=MASK_32_BIT) Funkcija u novoj poruci na mestu definisanom sa <i>offset</i> izvršava operaciju AND između vrednosti na datoj memorijskoj lokaciji i vrednosti <i>mask</i> a zatim izvršava OR sa vrednošću parametra <i>param</i> . |
| void | SetBitParamDWordBasic (DWORD param, uint32 offset, uint32 mask=MASK_32_BIT) Funkcija u novoj poruci na mestu definisanom sa <i>offset</i> izvršava operaciju AND između vrednosti na datoj memorijskoj lokaciji i vrednosti <i>mask</i> a zatim izvršava OR sa vrednošću parametra <i>param</i> . |
| void | SetBitParamWord (WORD param, uint32 offset, uint32 mask=MASK_32_BIT) Funkcija u novoj poruci na mestu definisanom sa <i>offset</i> izvršava operaciju AND između vrednosti na datoj memorijskoj lokaciji i vrednosti <i>mask</i> a zatim izvršava OR sa vrednošću parametra <i>param</i> . |
| inline void | SetCallId () Dodeljuje vrednost atributu koji predstavlja jedinstveni identifikator trenutnog poziva. |
| inline void | SetCallId (uint32 id) Dodeljuje vrednost atributu koji predstavlja jedinstveni identifikator trenutnog poziva. |
| inline void | SetCallIdFromMsg () Funkcija se koristi da atributu za praćenje trenutnog poziva dodeli vrednost koja je dobijena u primljenoj poruci. |
| virtual void | SetDefaultFSMData () Inicijalizuje attribute na odgovarajuće vrednosti pre početka rada automata. |
| virtual void | SetDefaultHeader (uint8 infoCoding) Definiše različita zaglavlja u zavisnosti od tipa kodiranja poruke. |
| inline void | SetGroup (uint8 id) Dodeljuje novu vrednost atributu koji identifikuje grupu kojoj automat pripada. |
| virtual void | SetInitialState () Funkcija postavlja stanje automata na inicijalno stanje. |
| static void | SetKernelObjects (TPostOffice *postOffice, TBuffers *buffers, CTimer *timer) Koristi se da kernelu pridruži sistemске objekte koji su jedinstveni za sve automate. |
| inline void | SetLeftMbx (uint8 mbx) Postavlja vrednost atributa za poštansko sanduče levog automata. |
| inline void | SetLeftAutomate (uint8 automate) Postavlja vrednost atributa koji definiše tip levog automata u odnosu na dati automat. |

| | |
|-------------|--|
| inline void | <u>SetLeftObject</u> (uint8 group) Postavlja vrednost atributa koji predstavlja grupu kojoj pripada levi automat. |
| inline void | <u>SetLeftObjectId</u> (uint32 id) Postavlja vrednost atributa za jedinstveni identifikator levog automata. |
| static void | <u>SetLogInterface</u> (LogInterface *loggingObject) Definiše koji će objekat za vođenje evidencije o toku rada rada sistema automata. |
| inline void | <u>SendMessage</u> (uint8 mbxId) Funkcija se koristi za slanje nove poruke u željeno poštansko sanduče. |
| inline void | <u>SendMessage</u> (uint8 mbxId, uint8 *msg) Koristi se za slanje poruke definisane parametrom <i>msg</i> u željeno poštansko sanduče. |
| void | <u>SetMessageFromData</u> () Funkcija u zaglavlje nove poruke postavlja podatke o automatu koji šalje poruku. |
| inline void | <u>SetMsgCallId</u> (uint32 id) Koristi se da postavi jedinstveni identifikacioni broj koji identifikuje jedan poziv. |
| inline void | <u>SetMsgCallId</u> (uint32 id, uint8 *msg) Koristi se da postavi jedinstveni identifikacioni broj koji identifikuje jedan poziv. |
| inline void | <u>SetMsgCode</u> (uint16 code) Funkcija se koristi za postavljanje koda poruke u novu poruku. |
| inline void | <u>SetMsgCode</u> (uint16 code, uint8 *msg) Koristi se za postavljanje koda poruke u poruku koja se nalazi u baferu. |
| inline void | <u>SetMsgFromAutomate</u> (uint8 from) Funkcija se koristi da u novoj poruci definiše automat koji je poslao poruku. |
| inline void | <u>SetMsgFromAutomate</u> (uint8 from, uint8 *msg) Koristi se da u poruci koja se nalazi u baferu definiše automat koji je poslao poruku. |
| inline void | <u>SetMsgFromGroup</u> (uint8 from) U novoj poruci definiše grupu kojoj pripada automat koji je poslao poruku. |
| inline void | <u>SetMsgFromGroup</u> (uint8 from, uint8 *msg) U poruci koja se nalazi u baferu definiše grupu automata koji je poslao poruku. |
| inline void | <u>SetMsgInfoCoding</u> (uint8 codingType) Postavlja u zaglavlju nove poruke vrednost kojom se definiše tip kodiranja poruke. |
| inline void | <u>SetMsgInfoCoding</u> (uint8 codingType, uint8 *msg) Funkcija postavlja u zaglavlju poruke, koja se nalazi u baferu datom sa parametrom <i>msg</i> , vrednost kojom se definiše tip kodiranja poruke. |
| inline void | <u>SetMsgInfoLength</u> (uint16 length) Funkcija postavlja u zaglavlje poruke dužinu informacionog dela poruke. |
| inline void | <u>SetMsgInfoLength</u> (uint16 length, uint8 *msg) Funkcijom se u zaglavlju poruke koja se nalazi u baferu postavlja vrednost za dužinu informacionog dela poruke. |
| inline void | <u>SetMsgObjectNumberFrom</u> (uint32 from) Koristi da u poruci postavi jedinstveni identifikator automata koji je poslao poruku. |
| inline void | <u>SetMsgObjectNumberFrom</u> (uint32 from, uint8 *msg) Koristi se da u poruci koja se nalazi u baferu postavi jedinstveni identifikator automata koji je poslao poruku. |
| inline void | <u>SetMsgObjectNumberTo</u> (uint32 to) Koristi se da u zaglavlju poruke postavi identifikator automata kojem se šalje poruka. |
| inline void | <u>SetMsgObjectNumberTo</u> (uint32 to, uint8 *msg) Koristi se da u zaglavlju poruke koja se nalazi u baferu postavi identifikator automata kojem se šalje poruka. |
| inline void | <u>SetMsgToAutomate</u> (uint8 to) Koristi se da u zaglavlju nove poruke postavi tip automata kojem se šalje poruka. |
| inline void | <u>SetMsgToAutomate</u> (uint8 to, uint8 *msg) Koristi se da u zaglavlju poruke koja se nalazi u baferu postavi tip automata kojem se šalje poruka. |
| inline void | <u>SetMsgToGroup</u> (uint8 to) Koristi se da u zaglavlju poruke postavi grupu automata kojem se šalje poruka. |

| | |
|-------------|--|
| inline void | SetMsgToGroup (uint8 to, uint8 *msg) Koristi se da u zaglavlju poruke koja se nalazi u baferu postavi grupu kojoj pripada automat kojem se šalje poruka. |
| void | SendMessageLeft () Funkcija šalje novu poruku logički levom automatu. |
| void | SendMessageRight () Funkcija šalje novu poruku logički desnom automatu. |
| inline void | SetNewMessage (uint8 *msg) Funkcija dodeljuje atributu NewMessage adresa bafera u kojem je već formatirana nova poruka ili u kojem treba da se formira nova poruka. |
| inline void | SetObjectId (uint32 id) Funkcija dodeljuje jedinstveni identifikacioni broj automatu. |
| inline void | SetRightMbx (uint8 mbx) Postavlja vrednost atributa za poštansko sanduče desnog automata. |
| inline void | SetRightAutomate (uint8 automate) Postavlja vrednost atributa koji definiše tip desnog automata. |
| inline void | SetRightObject (uint8 group) Postavlja vrednost atributa koji predstavlja grupu kojoj pripada desni automat. |
| inline void | SetRightObjectId (uint32 id) Postavlja vrednost atributa za jedinstveni identifikator logički desnog automata. |
| inline void | SetState (uint8 state) Koristi se da postavi stanje automata u novo stanje definisano parametrom funkcije. |
| void | StartTimer (uint16 tmrId) Pokreće vremensku kontrolu čiji je identifikator dat parametrom <i>tmrId</i> . |
| void | StopTimer (uint16 tmrId) Funkcija zaustavlja vremensku kontrolu čiji je identifikator dat parametrom <i>tmrId</i> . |
| static void | SysClearLogFlag () Funkcija zaustavlja vođenje evidencije poruka koje razmenjuju automati. |
| static void | SysStartAll () Funkcija pokreće vođenje evidencije poruka koje razmenjuju automati. |

Konstruktor klase NetFSM

[NetFSM](#)(uint16 numOfTimers = DEFAULT_TIMER_NO, uint16 numOfState = DEFAULT_STATE_NO, uint16 maxNumOfProceduresPerState = DEFAULT_PROCEDURE_NO_PER_STATE, bool getMemory = true)
Inicijalizuje instancu datog automata i strukture podataka koje su potrebne za njegov pravilan rad.

Metode klase NetFSM

| | |
|----------------|---|
| virtual void | convertFSMToNetMessage () Konvertuje format poruke koji automati razmenjuju u okviru sistema automata u format poruke koja se šalje preko TCP/IP protokola. |
| virtual uint16 | convertNetToFSMMessage () Konvertuje format poruke koji je primljen od strane TCP/IP protokola u format poruke koji automati razmenjuju u okviru sistema automata. |
| void | establishConnection () Funkcija se koristi za uspostavljanje veze između dva sistema automata. |
| virtual uint8 | getProtocolInfoCoding () Funkcija vraća vrednost tipa kodiranja poruke. |
| void | sendToTCP () Funkcija šalje poruku iz jednog sistema automata u drugi sistem automata. |

FSMSystem

```
FSMSystem(    uint8 numOfAutomates,  
             uint8 numberOfMbx )
```

Konstruktor FSMSystem() ima osnovni zadatak da napravi objekat koji predstavlja sistem automata i da inicijalizuje strukture podataka koje su potrebne za njegov pravilan rad.

Parametri:

numOfAutomates - Broj različitih vrsta automata koji će biti uključeni u sistem automata.
numberOfMbx - Broj poštanskih sandučića preko kojih automati mogu da razmenjuju poruke.

Napomena:

Preporučuje se da za svaki tip automata bude rezervisano po jedno poštansko sanduče iako je moguće da svi automati komuniciraju preko samo jednog poštanskog sandučeta.

Nazad na [API funkcije](#)

Add

```
void Add(    ptrFiniteStateMachine object,  
            uint8 automateType,  
            uint32 numOfObjects,  
            bool useFreeList = false )
```

Ova funkcija se koristi prilikom inicijalizacije sistema automata. Svaki put kada se u sistem automata uključuje novi tip automata ovom funkcijom se u sistem automata dodaje prvi automat pri čemu se definiše tip automata, ukupan broj automata ovog tipa koji će biti uključen u sistem automata kao i informacija o tome da li će se ovi automati koristiti preko liste slobodnih automata.

Parametri:

object - Adresa prvog automata datog tipa koji se dodaje u sistem automata.
automateType - Novi tip automata koji se dodaje u sistem automata.
numOfObjects - Ukupan broj automata ovog tipa koji će biti uključen u sistem automata.
useFreeList - Informacija da li se automati ovog tipa koriste preko liste slobodnih automata.

Napomena:

Za svaki tip automata ova funkcija mora biti pozvana za prvi automat koji se dodaje u sistem automata zato što se u okviru ove funkcije inicijalizuju interne strukture podataka sistema automata. Svi ostali automati nakon prvog se dodaju drugom funkcijom [Add\(\)](#) koja ima samo dva parametra. Ako se koristi lista slobodnih automata tada se poruka može poslati slobodnom automatu tako što će se umesto identifikacionog broja automata kojem se šalje poruka staviti vrednost -1, vidi [SetMsgObjectNumberTo\(\)](#).

Nazad na [API funkcije](#)

Add

```
void Add(    ptrFiniteStateMachine object,  
            uint8 automateType )
```

Ova funkcija se koristi prilikom inicijalizacije sistema automata. Svi automati osim prvog od svakog tipa se uključuju u sistem automata ovom funkcijom.

Parametri:

object - Adresa automata datog tipa koji se dodaje u sistem automata.
automateType - Tip automata koji se dodaje u sistem automata.

Nazad na [API funkcije](#)

InitKernel

```
void InitKernel(      uint8 buffClassNo,  
                     uint32 *buffersCount,  
                     uint32 *buffersLength,  
                     uint8 numOfMbxes=0,  
                     TimerResolutionEnum timerRes = Timer1s )
```

Funkcijom InitKernel() se inicijalizuju elementi kernela za rukovanje porukama, memorijom, i vremenskim kontrolama. Ova funkcija mora biti pozvana pre pokretanja sistema automata u rad.

Parametri:

buffClassNo - Predstavlja broj tipova bafera koji su na raspolaganju sistemu automata.
buffersCount - Adresa niza koji sadži za svaki tip bafera broj bafera.
buffersLength - Adresa niza koji sadži za svaki tip bafera veličinu bafera.
numOfMbxes - Broj poštanskih sandučića koji će se inicijalizovati u okviru sistema automata.
timerRes - Rezolucija vremenskih kontrola (najmanja jedinica vremena).

Napomena:

Funkcija InitKernel se mora pozvati nakon što se naprave sve instance automata ali pre nego što se pokrene sistem automata. Rezolucija vremenskih kontrola je po definiciji 1 sekunda.

Nazad na [API funkcije](#)

Remove

```
void Remove( uint8 automateType )
```

Funkcija Remove() uklanja sve automate datog tipa iz sistema automata.

Parametri:

automateType - Tip automata koji treba da ukloni iz sistema automata.

Napomena:

Nakon što se automati isključe iz sistema automata oslobađa se i zauzeta memorija za interne strukture podataka koje su korišćene za dati tip automata.

Nazad na [API funkcije](#)

Remove

```
ptrFiniteStateMachine Remove(      uint8 automateType,
```



```
uint32 object )
```

Funkcija Remove() uklanja automat sa identifikacionim brojem koji je smešten u parametru *object* datog tipa iz sistema automata.

Parametri:

automateType - Tip automata koji treba da isključi iz sistema automata.
object - Identifikacioni broj automata koji treba da se isključi iz sistema automata.

Povratna Vrednost:

Adresa objekta koji predstavlja instancu automata koji je isključen iz sistema automata.

Nazad na [API funkcije](#)

Start

```
virtual void Start()
```

Start() je funkcija kojom se pokreće rad sistema automata. U okviru ove funkcije sistem automata po utvrđenom redosledu preuzima poruke iz poštanskih sandučića i prosleđuje ih odgovarajućim automatima na obradu.

Napomena:

Obrada poruka se izvršava u petlji koja se kontroliše internim atributom SystemWorking. Preporučeni način implementacije ove funkcije je prikazan u poglavlju [2.1.2](#).

Nazad na [API funkcije](#)

StopSystem

```
void StopSystem()
```

Funkcija StopSystem() se koristi za zaustavljanje rada sistema automata. Ovom funkcijom se internom atributu SystemWorking dodeljuje odgovarajuća vrednost koja će izazvati prekid rada sistema automata.

Napomena:

Ako je funkcija [Start\(\)](#) pokrenuta u posebnoj programskoj niti tada će se završiti i rad odgovarajuće programske niti.

Nazad na [API funkcije](#)

FSMSystemWithTCP

```
FSMSystemWithTCP(    uint8 numOfAutomates,  
                     uint8 numberOfMbx )
```

Konstruktor FSMSystemWithTCP() ima osnovni zadatak da napravi objekat koji predstavlja sistem automata koji ima podršku za rad sa TCP/IP protokolom i da inicijalizuje strukture podataka koje su potrebne za njegov pravilan rad.

Parametri:

`numOfAutomates` - Broj različitih vrsta automata koji će biti uključeni u sistem automata.
`numberOfMbx` - Broj poštanskih sandučića preko kojih automati mogu da razmenjuju poruke.

Napomena:

Preporučuje se da za svaki tip automata bude rezervisano po jedno poštansko sanduče iako je moguće da svi automati komuniciraju preko samo jednog poštanskog sandučeta.

Nazad na [API funkcije](#)

InitTCPServer

```
int InitTCPServer( uint16 port,
                  uint8 automateType,
                  char *ipAddress = 0,
                  unsigned char *parm = 0,
                  int length = 0 )
```

Funkcija `InitTCPServer()` se koristi da inicijalizuje server koji čeka da stigne zahtev za vezom od drugog sistema automata. U slučaju dolaska zahteva usluživač izdvaja prvi slobodan automat navedenog tipa (*automateType*) i njemu prosleđuje parametre (*parm*) zajedno sa novoostvarenom vezom. Dalja komunikacija se obavlja sa datim automatom, tj. usluživač je isključen iz komunikacije.

Parametri:

`port` - Parametar čija vrednost predstavlja port na kojem će server očekivati zahteve za vezom.
`automateType` - Parametar čija vrednost predstavlja tip automata koji je tipa `NetFSM` i kojima usluživač prepušta novoostvarenu vezu. Dalju komunikaciju sa udaljenom stranom obavlja automat kome je prepuštena veza.
`ipAddress` - Parametar čija vrednost predstavlja IP adresu na kojoj će server da čeka vezu.
`parm` - Pokazivač na parametre koji se prosleđuju tipu automata navedenom kao *automateType*.
`length` - Predstavlja dužinu u bajtima parametara koji se prosleđuju tipu automata navedenom kao *automateType*.

Povratna Vrednost:

Funkcija vraća vrednost 0 ako je uspešno pokrenut server koji očekuje uspostavu veze u suprotnom vraća vrednost -1.

Nazad na [API funkcije](#)

FiniteStateMachine

```
FiniteStateMachine( uint16 numOfTimers = DEFAULT_TIMER_NO,
                   uint16 numOfState = DEFAULT_STATE_NO,
                   uint16 maxNumOfProceduresPerState = DEFAULT_PROCEDURE_NO_PER_STATE,
                   bool getMemory = true );
```

Konstruktor `FiniteStateMachine()` ima osnovni zadatak da napravi instancu datog automata i da inicijalizuje strukture podataka koje su potrebne za njegov pravilan rad.

Parametri:

numOfTimers - Broj vremenskih kontrola koje će koristiti ovaj automat.
numOfState - Broj stanja koje imaju automati ovog tipa.
maxNumOfProceduresPerState - Maksimalan broj funkcija prelaza za jedno stanje automata.
getMemory - Indikator na osnovu kog konstruktor rezerviše memoriju za stanja i funkcije prelaza

Napomena:

Konstruktor je moguće pozvati bez parametara, tada će se automat inicijalizovati sa predefinisanim vrednostima. Parametar getMemory se koristi kada je potrebno izvršiti optimizaciju zauzimanja slobodne memorije za tabelu sa funkcijama prelaza.

Nazad na [API funkcije](#)

AddParam

```
uint8 *AddParam(      uint16 paramCode,  
                      uint32 paramLength,  
                      uint8 *param )
```

Funkcija AddParam() se koristi za dodavanje parametra u novu poruku koja tek treba da se pošalje. Parametri su u poruci sortirani po rastućoj vrednosti koda parametra.

Parametri:

paramCode - Vrednost koda parametra.
paramLength - Dužina informacionog dela parametra koji treba da se doda u poruku.
param - Adresa na kojoj je smešten parametar koji će biti iskopiran u bafer poruke.

Povratna Vrednost:

Funkcija vraća adresu bafera u koji je smeštena nova poruka.

Napomena:

Funkcija omogućava da se u poruku smesti parametar proizvoljne dužine sa ograničenjem da dužina parametra ne može da bude veća od maksimalne dužine koja je dopuštena za datu vrstu kodiranja poruke (za StandardMessage najviše 256 bajtova).

Nazad na [API funkcije](#)

AddParamByte

```
uint8 *AddParamByte(      uint16 paramCode,  
                           BYTE param )
```

Funkcija AddParamByte() se koristi za dodavanje parametra veličine jednog bajta u novu poruku koja tek treba da se pošalje. Parametri su u poruci sortirani po rastućoj vrednosti koda parametra.

Parametri:

paramCode - Vrednost koda parametra
param - Ovaj parametar sadrži vrednost veličine jednog bajta koja će biti kopirana u poruku.

Povratna Vrednost:

Funkcija vraća adresu bafera u koji je smeštena nova poruka.

Nazad na [API funkcije](#)

AddParamDWord

```
uint8 *AddParamDWord(      uint16 paramCode,  
                           DWORD param )
```

Funkcija AddParamDWord() se koristi za dodavanje parametra veličine četiri bajta u novu poruku koja tek treba da se pošalje. Parametri su u poruci sortirani po rastućoj vrednosti koda parametra.

Parametri:

paramCode - Vrednost koda parametra.

param - Ovaj parametar sadrži vrednost veličine četiri bajta koja će biti kopirana u poruku.

Povratna Vrednost:

Funkcija vraća adresu bafera u koji je smeštena nova poruka.

Nazad na [API funkcije](#)

AddParamWord

```
uint8 *AddParamWord(      uint16 paramCode,  
                           WORD param )
```

Funkcija AddParamWord() se koristi za dodavanje parametra veličine dva bajta u novu poruku koja tek treba da se pošalje. Parametri su u poruci sortirani po rastućoj vrednosti koda parametra.

Parametri:

paramCode - Vrednost koda parametra.

param - Ovaj parametar sadrži vrednost veličine dva bajta koja će biti kopirana u poruku.

Povratna Vrednost:

Funkcija vraća adresu bafera u koji je smeštena nova poruka.

Nazad na [API funkcije](#)

CheckBufferSize

```
virtual void CheckBufferSize( uint32 paramLength )
```

Funkcija CheckBufferSize() se koristi da bi se obezbedio bafer dovoljno velik da u njega može da se smesti dati parametar. Parametri su u poruci sortirani po rastućoj vrednosti koda parametra.

Parametri:

paramLength - Dužina parametra koji se želi dodati u poruku u bajtovima.

Napomena:

Prvo se proverava da li dati parametar može da se smesti u bafer u kojem se trenutno nalazi nova poruka koja tek treba da se pošalje. A ako ne može uzima se novi dovoljno velik bafer i u njega se kopira sadržaj starog bafera.

Nazad na [API funkcije](#)

ClearMessage

```
virtual void ClearMessage()
```

Funkcija ClearMessage() vraća nazad programskom jezgru bafer primljene poruke i dodeljuje vrednost NULL pokazivaču na primljenu poruku. Ako je programski kod iskompajliran da radi u modu za otkrivanje grešaka izvršiće se dodatna provera da li je pokazivač na novu poruku jednak NULL.

Nazad na [API funkcije](#)

CopyMessage

```
virtual void CopyMessage()
```

Funkcija CopyMessage() se koristi da napravi kopiju primljene poruke. Adresa bafera u koji je smeštena kopija primljene poruke dodeljena je pokazivaču na novu poruku.

Napomena:

Ako je pokazivač na novu poruku pre poziva ove funkcije imao vrednost adrese nekog bafera (pripremana je neka druga poruka za slanje) ovaj bafer će biti vraćen kernelu pre preuzimanja novog bafera i kopiranja sadržaja primljene poruke.

Nazad na [API funkcije](#)

CopyMessage

```
virtual void CopyMessage( uint8 *msg )
```

Funkcija CopyMessage() se koristi da napravi kopiju poruke koja je smeštena u baferu čija je adresa data parametrom. Adresa bafera u koji je smeštena kopija date poruke dodeljena je pokazivaču na novu poruku.

Parametri:

msg - Adresa bafera u kojem se nalazi poruka koju želimo da iskopiramo.

Napomena:

Pokazivač na novu poruku pre poziva ove funkcije mora imati vrednost NULL.

Nazad na [API funkcije](#)

CopyMessageInfo

```
virtual void CopyMessageInfo(    uint8 infoCoding,  
                                uint16 lengthCorrection = 0 )
```

Funkcija CopyMessageInfo() se koristi da kopira informacioni deo primljene poruke (bez zaglavlja) u novi bafer čija je adresa dodeljena pokazivaču na novu poruku.

Parametri:

`infoCoding` - Informacija o vrsti kodiranja koja je korišćena za formatiranje primljene poruke.
`lengthCorrection` - Korekcija dužine poruke. Zavisi od vrste kodiranja.

Nazad na [API funkcije](#)

Discard

```
virtual void Discard( uint8* buff )
```

Funkcijom `Discard()` se briše poruka koja je smeštena u datom baferu i bafer se vraća nazad kernelu.

Parametri:

`buff` - Adresa u memoriji gde je smešten bafer koji želimo da vratimo kernelu.

Nazad na [API funkcije](#)

DoNothing

```
void DoNothing()
```

Funkcija `DoNothing()` je prazna funkcija. Ova funkcija se poziva kada u automat stigne ne očekivana poruka (za dato stanje i dati kod poruke nije definisana funkcija obrade).

Napomena:

Ovu funkciju je moguće redefinisati funkcijom [InitUnexpectedEventProc\(\)](#) ako je potrebno da se promeni predefinisano ponašanje automata na ne očekivane poruke.

Nazad na [API funkcije](#)

FreeFSM

```
void FreeFSM()
```

Funkcija `FreeFSM()` se koristi da javi sistemu automata da je ova instanca automata završila svoju obradu i da je automat slobodan. Ako je za datu vrstu automata prilikom dodavanja u sistem automata naznačeno da će se koristiti lista slobodnih automata ovom funkcijom dati automat će biti vraćen u listu slobodnih automata.

Napomena:

Ako je prilikom inicijalizacije sistema automata naznačeno da se neće koristiti lista slobodnih automata ova funkcija nema nikakvog efekta.

Nazad na [API funkcije](#)

GetAutomate

```
virtual uint8 GetAutomate() = 0
```

Funkcija `GetAutomate()` vraća vrednost koja identifikuje tip automata.

Povratna Vrednost:

Identifikacioni broj koji je dodeljen datom tipu automata.

Napomena:

Ova funkcija je prava virtuelna funkcija koja mora da se definiše u klasi koja implementira dati automat. Uobičajena je procedura da funkcija vraća konstantnu vrednost koja je deklarirana u datoteci u kojoj su definisani identifikacioni brojevi svih automata koji su uključeni u sistem automata.

Nazad na [API funkcije](#)

GetBitParamByteBasic

```
uint8 GetBitParamByteBasic(    uint32 offset,  
                               uint32 mask=MASK_32_BIT )
```

Funkcija `GetBitParamByteBasic()` vraća vrednost jednog bajta koja se nalazi u primljenoj poruci na udaljenosti od početka poruke za vrednost parametra *offset* nad kojom je izvršena binarna operacija AND sa vrednošću parametra *mask*.

Parametri:

offset - Predstavlja indeks u odnosu na početak poruke.

mask - Vrednost kojom se maskira dati bajt u poruci da bi se dobila vrednost ove funkcije.

Povratna Vrednost:

Rezultat veličine jednog bajta binarne operacije AND između vrednosti koja se u primljenoj poruci nalazi na mestu *offset* i vrednosti parametra *mask*.

Napomena:

U zavisnosti od vrednosti parametra *mask* moguće je testirati vrednost jednog ili više bita istovremeno u okviru bajta koji se nalazi na mestu definisanom sa *offset* u odnosu na početak primljene poruke.

Nazad na [API funkcije](#)

GetBitParamWordBasic

```
uint16 GetBitParamWordBasic(    uint32 offset,  
                                 uint32 mask=MASK_32_BIT )
```

Funkcija `GetBitParamWordBasic()` vraća vrednost dva bajta (Word) koja se nalazi u primljenoj poruci na udaljenosti od početka poruke za vrednost parametra *offset* nad kojom je izvršena binarna operacija AND sa vrednošću parametra *mask*.

Parametri:

offset - Predstavlja indeks u odnosu na početak poruke.

mask - Vrednost kojom se maskiraju dva bajta u poruci da bi se dobila vrednost ove funkcije.

Povratna Vrednost:

Rezultat veličine dva bajta binarne operacije AND između vrednosti koja se u primljenoj poruci nalazi na mestu *offset* i vrednosti parametra *mask*.

Napomena:

U zavisnosti od vrednosti parametra *mask* moguće je testirati vrednost jednog ili više bita istovremeno u okviru dva bajta koji se nalaze na mestu definisanom sa *offset* u odnosu na početak primljene poruke.

Nazad na [API funkcije](#)

GetBitParamDWordBasic

```
uint32 GetBitParamDWordBasic(    uint32 offset,  
                                uint32 mask=MASK_32_BIT )
```

Funkcija `GetBitParamDWordBasic()` vraća vrednost četiri bajta (DWord) koja se nalazi u primljenoj poruci na udaljenosti od početka poruke za vrednost parametra *offset* nad kojom je izvršena binarna operacija AND sa vrednošću parametra *mask*.

Parametri:

offset - Predstavlja indeks u odnosu na početak poruke.
mask - Vrednost kojom se maskiraju četiri bajta u poruci da bi se dobila vrednost ove funkcije.

Povratna Vrednost:

Rezultat veličine četiri bajta binarne operacije AND između vrednosti koja se u primljenoj poruci nalazi na mestu *offset* i vrednosti parametra *mask*.

Napomena:

U zavisnosti od vrednosti parametra *mask* moguće je testirati vrednost jednog ili više bita istovremeno u okviru četiri bajta koji se nalaze na mestu definisanom sa *offset* u odnosu na početak primljene poruke.

Nazad na [API funkcije](#)

GetBuffer

```
virtual uint8 *GetBuffer( uint32 length )
```

Funkcija vraća adresu bafera koji je veći ili jednak od vrednosti zadate parametrom *length*.

Parametri:

length - Vrednost ovog parametra predstavlja minimalanu veličinu bafera u bajtovima.

Povratna Vrednost:

Adresa bafera koji je dovoljno velik da smesti broj bajtova koji je zadat parametrom *length*.

Napomena:

Pošto je veličina bafera definisana preko diskretnih vrednosti koje se definišu prilikom inicijalizacije kernela ova funkcija uvek vraća najmanji bafer u koji je moguće smestiti onoliko

bajtova koliko je definisano vrednošću parametra `length`. Ako nema slobodnih bafera ova funkcija će generisati izuzetak koji će ukoliko ne bude obrađen izazvati kraj rada sistema automata.

Nazad na [API funkcije](#)

GetBufferLength

```
uint32 GetBufferLength( uint8 *buff )
```

Funkcija `GetBufferLength()` vraća veličinu bafera u bajtovima. Bafer se nalazi na adresi koja je data kao parametar ove funkcije kroz vrednost pokazivača `buff`.

Parametri:

`buff` - Pokazivač čija vrednost predstavlja adresu bafera čiju veličinu tražimo.

Povratna Vrednost:

Funkcija vraća veličinu bafera u bajtovima.

Nazad na [API funkcije](#)

GetCallId

```
virtual inline uint32 GetCallId()
```

Funkcija `GetCallId()` vraća trenutnu vrednost atributa `CallId` koji se koristi za praćenje toka poziva u kojem automat trenutno učestvuje.

Povratna Vrednost:

Funkcija vraća vrednost atributa `CallId`.

Nazad na [API funkcije](#)

GetCount

```
uint32 GetCount( uint8 mbx )
```

Funkcija `GetCount()` vraća broj poruka koje čekaju na obradu u poštanskom sandučetu sa rednim brojem koji je dat parametrom `mbx`.

Parametri:

`mbx` - Vrednost ovog parametra je broj poštanskog sandučeta čiji se broj poruka traži.

Povratna Vrednost:

Funkcija vraća broj poruka koje čekaju na obradu u datom poštanskom sandučetu.

Nazad na [API funkcije](#)

GetGroup

```
virtual uint8 GetGroup()
```

Opis:

Funkcija GetGroup() vraća broj koji identifikuje grupu kojoj ovaj automat pripada.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje grupu kojoj ovaj automat pripada.

Nazad na [API funkcije](#)

GetInitialState

```
virtual uint8 GetInitialState()
```

Funkcija GetInitialState() vraća broj koji identifikuje inicijalno stanje u kojem se automat nalazi.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje inicijalno stanje u kojem se automat nalazi.

Napomena:

Predefinisana vrednost inicijalnog stanja je broj 0.

Nazad na [API funkcije](#)

GetLeftMbx

```
virtual inline uint8 GetLeftMbx()
```

Funkcija GetLeftMbx() vraća broj koji identifikuje poštansko sanduče automata koji se na logičkom nivou nalazi levo od automata koji je pozvao ovu funkciju.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje poštansko sanduče automata koji se logički posmatrano nalazi levo od trenutnog automata.

Nazad na [API funkcije](#)

GetLeftAutomate

```
virtual inline uint8 GetLeftAutomate()
```

Funkcija GetLeftAutomate() vraća broj koji identifikuje tip automata koji se na logičkom nivou nalazi levo od automata koji je pozvao ovu funkciju.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje tip automata koji se logički posmatrano nalazi levo od trenutnog automata.

Nazad na [API funkcije](#)

GetLeftGroup

```
virtual inline uint8 GetLeftGroup()
```

Funkcija GetLeftGroup() vraća broj koji identifikuje grupu kojoj pripadaju automati koji se na logičkom nivou nalaze levo od automata koji je pozvao ovu funkciju.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje grupu automata koji se logički posmatrano nalazi levo od trenutnog automata.

Nazad na [API funkcije](#)

GetLeftObjectId

```
virtual inline uint32 GetLeftObjectId()
```

Funkcija GetLeftObjectId() vraća broj koji identifikuje automat koji se na logičkom nivou nalazi levo od automata koji je pozvao ovu funkciju.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje automat koji se logički posmatrano nalazi levo od trenutnog automata.

Nazad na [API funkcije](#)

GetMbxId

```
virtual uint8 GetMbxId() = 0
```

Funkcija GetMbxId() vraća broj poštanskog sandučeta u koji stižu poruke za dati automat.

Povratna Vrednost:

Funkcija vraća broj poštanskog sandučeta automata.

Napomena:

Ova funkcija je prava virtuelna funkcija koja mora da se definiše u klasi koja implementira dati automat. Uobičajena je procedura da funkcija vraća konstantnu vrednost koja je deklarirana u datoteci u kojoj su definisani identifikacioni brojevi svih automata koji su uključeni u sistem automata.

Nazad na [API funkcije](#)

GetMessageInterface

```
virtual MessageInterface *GetMessageInterface( uint32 id ) = 0
```

Funkcijom GetMessageInterface() preuzima se od automata objekat koji implementira klasu MessageInterface, a kojim se definiše kako se kodiraju poruke za dati automat u okviru sistema automata.

Parametri:

`id` - Ovaj parametar predstavlja identifikator koji definiše tip kodiranja poruka.

Povratna Vrednost:

Funkcija vraća adresu objekta kojim se rukuje porukama za dati tip kodiranja.

Napomena:

Ova funkcija je prava virtuelna funkcija koja se mora implementirati u okviru svake klase automata. Id sa vrednošću 0 je rezervisan za kodiranje poruka u formatu StandardMessage koji predstavlja osnovni tip poruke u okviru sistema automata.

Nazad na [API funkcije](#)

GetMsg

```
uint8* GetMsg()
```

Funkcija GetMsg() uzima prvu sledeću poruku iz poštanskog sandučeta koje je definisano za prijem poruka ovog automata.

Povratna Vrednost:

Funkcija vraća adresu bafera u koji je smeštena sledeća poruka ili vrednost NULL ako je poštansko sanduče prazno (nema novih poruka).

Nazad na [API funkcije](#)

GetMsg

```
static uint8* GetMsg( uint8 mbx )
```

Funkcija GetMsg() uzima prvu sledeću poruku iz poštanskog sandučeta koje je definisano vrednošću parametra *mbx*.

Parametri:

`mbx` - Broj poštanskog sandučeta iz kog će ova funkcija uzeti prvu sledeću poruku.

Povratna Vrednost:

Funkcija vraća adresu bafera u koji je smeštena sledeća poruka ili vrednost NULL ako je poštansko sanduče prazno (nema novih poruka).

Napomena:

Iako je ova funkcija definisana kao statička poziv ove funkcije pre nego što se inicijalizuju kernel i sistem automata imaće za rezultat ne definisan rad programa.

Nazad na [API funkcije](#)

GetMsgCallId

```
inline uint32 GetMsgCallId()
```

Funkcija `GetMsgCallId()` vraća jedinstveni identifikacioni broj poziva koji je isti za sve poruke koje se razmenjuju u toku jednog poziva.

Povratna Vrednost:

Funkcija vraća jedinstveni identifikacioni broj poziva.

Nazad na [API funkcije](#)

GetMsgCode

```
inline uint16 GetMsgCode()
```

Funkcija `GetMsgCode()` vraća kod poruke koji se preuzima iz zaglavlja primljene poruke.

Povratna Vrednost:

Funkcija vraća kod priljene poruke.

Nazad na [API funkcije](#)

GetMsgFromAutomate

```
inline uint8 GetMsgFromAutomate()
```

Funkcija `GetMsgFromAutomate()` vraća broj koji odgovara tipu automata koji je poslao poruku. Ova vrednost se preuzima iz zaglavlja primljene poruke.

Povratna Vrednost:

Funkcija vraća tip automata koji je poslao poruku.

Nazad na [API funkcije](#)

GetMsgFromGroup

```
inline uint8 GetMsgFromGroup()
```

Funkcija `GetMsgFromGroup()` vraća broj grupe kojoj pripada automat koji je poslao poruku. Ova vrednost se preuzima iz zaglavlja primljene poruke.

Povratna Vrednost:

Funkcija vraća identifikator grupe automata koji je poslao poruku.

Nazad na [API funkcije](#)

GetMsgInfoCoding

```
inline uint8 GetMsgInfoCoding()
```

Funkcija `GetMsgInfoCoding()` vraća broj koji identifikuje tip kodiranja poruke.

Povratna Vrednost:

Funkcija vraća broj kojim se identifikuje tip kodiranja kojim je kodirana primljena poruka.

Nazad na [API funkcije](#)

GetMsgInfoLength

```
inline uint16 GetMsgInfoLength()
```

Funkcija `GetMsgInfoLength()` vraća dužinu informacionog dela primljene poruke.

Povratna Vrednost:

Funkcija vraća dužinu informacionog dela poruke.

Napomena:

U ovu dužinu nije uključena veličina zaglavlja primljene poruke. Ukupna dužina primljene poruke je jednaka zbiru dužine zaglavlja i dužine informacionog dela poruke.

Nazad na [API funkcije](#)

GetMsgInfoLength

```
inline uint16 GetMsgInfoLength( uint8 *msg )
```

Funkcija `GetMsgInfoLength()` vraća dužinu informacionog dela poruke koja se nalazi u baferu čija je adresa data parametrom *msg*.

Parametri:

msg - Pokazivač u kojem se nalazi adresa bafera u kojem se nalazi poruka.

Povratna Vrednost:

Funkcija vraća dužinu informacionog dela poruke.

Napomena:

U ovu dužinu nije uključena veličina zaglavlja date poruke. Ukupna dužina poruke je jednaka zbiru dužine zaglavlja i dužine informacionog dela poruke.

Nazad na [API funkcije](#)

GetMsgObjectNumberFrom

```
inline uint32 GetMsgObjectNumberFrom()
```

Funkcija `GetMsgObjectNumberFrom()` vraća identifikacioni broj automata koji je poslao poruku.

Povratna Vrednost:

Funkcija vraća identifikacioni broj automata koji je poslao poruku.

Napomena:

Ova vrednost se nalazi u zaglavlju primljene poruke.

Nazad na [API funkcije](#)

GetMsgObjectNumberTo

```
inline uint32 GetMsgObjectNumberTo()
```

Funkcija GetMsgObjectNumberTo() vraća identifikacioni broj automata kojem je poslata poruka.

Povratna Vrednost:

Funkcija vraća identifikacioni broj automata kojem je poslata poruka.

Napomena:

Ova vrednost se nalazi u zaglavlju primljene poruke.

Nazad na [API funkcije](#)

GetMsgToAutomate

```
inline uint8 GetMsgToAutomate()
```

Funkcija GetMsgToAutomate() vraća identifikacioni broj tipa automata kojem je poslata poruka.

Povratna Vrednost:

Funkcija vraća identifikacioni broj tipa automata kojem je poslata poruka.

Napomena:

Ova vrednost se nalazi u zaglavlju primljene poruke.

Nazad na [API funkcije](#)

GetMsgToGroup

```
inline uint8 GetMsgToGroup()
```

Funkcija GetMsgToGroup() vraća identifikacioni broj grupe kojoj automat kojem je poslata poruka pripada.

Povratna Vrednost:

Funkcija vraća identifikacioni broj grupe kojoj automat kojem je poslata poruka pripada.

Napomena:

Ova vrednost se nalazi u zaglavlju primljene poruke.

Nazad na [API funkcije](#)

GetNewMessage

```
inline uint8 *GetNewMessage()
```

Funkcija GetNewMessage() vraća adresu bafera u kojem se nalazi nova poruka.

Povratna Vrednost:

Funkcija vraća adresu bafera u kojem se nalazi nova poruka.

Napomena:

Ako nije rezervisan bafer za novu poruku tada će povratna vrednost biti NULL. Zauzimanje bafera za novu poruku se izvodi pozivom funkcije [PrepareNewMessage\(\)](#) ili [GetBuffer\(\)](#).

Nazad na [API funkcije](#)

GetNewMsgInfoLength

```
inline uint16 GetNewMsgInfoLength()
```

Funkcija GetNewMsgInfoLength() vraća dužinu informacionog dela nove poruke.

Povratna Vrednost:

Funkcija vraća dužinu informacionog dela nove poruke.

Napomena:

Ukoliko je vrednost pokazivača na novu poruku jednaka NULL poziv ove funkcije generisaće ne definisanu vrednost.

Nazad na [API funkcije](#)

GetNewMsgInfoCoding

```
inline uint8 GetNewMsgInfoCoding()
```

Funkcija GetNewMsgInfoCoding() vraća broj koji identifikuje vrstu kodiranja nove poruke.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje vrstu kodiranja nove poruke.

Napomena:

Ova vrednost se preuzima iz zaglavlja nove poruke.

Nazad na [API funkcije](#)

GetNextParam

```
uint8 *GetNextParam( uint16 paramCode )
```


Funkcija `GetNextParam()` vraća adresu na kojoj se nalazi sledeća instanca traženog parametara unutar primljene poruke.

Parametri:

`paramCode` - Predstavlja kod parametra koji se traži u primljenoj poruci.

Povratna Vrednost:

Funkcija vraća adresu na kojoj se nalazi traženi parametar ili vrednost `NULL` ako traženi parametar nije pronađen u primljenoj poruci.

Napomena:

Prva instanca svakog parametra se mora preuzeti odgovarajućom funkcijom (koja u svom nazivu nema `Next`) za ovu funkciju to je [GetParam\(\)](#). Upotreba `next` funkcija smanjuje vreme pretrage za parametrima unutar primljene poruke.

Nazad na [API funkcije](#)

GetNextParamByte

```
bool GetNextParamByte(    uint16 paramCode,
                          BYTE    &param )
```

Funkcija `GetNextParamByte()` vraća vrednost `TRUE` ako je pronađena sledeća instanca parametra zadata vrednošću `paramCode` i njegova vrednost dodeljena promenljivoj `param`. Ako je pretraga bila ne uspešna funkcija vraća vrednost `FALSE`.

Parametri:

`paramCode` - Predstavlja kod parametra koji se traži u primljenoj poruci.

`param` - Promenljiva u koju će se smestiti vrednost traženog parametra veličine jednog bajta.

Povratna Vrednost:

Funkcija vraća `TRUE` ako je traženi parametar pronađen u primljenoj poruci i njegova vrednost dodeljena promenljivoj `param` u suprotnom funkcija vraća vrednost `FALSE`.

Napomena:

Prva instanca svakog parametra se mora preuzeti odgovarajućom funkcijom (koja u svom nazivu nema `Next`) za ovu funkciju to je [GetParamByte\(\)](#). Upotreba `next` funkcija smanjuje vreme pretrage za parametrima unutar primljene poruke.

Nazad na [API funkcije](#)

GetNextParamDWord

```
bool GetNextParamDWord(    uint16 paramCode,
                           DWORD    &param )
```

Funkcija `GetNextParamDWord()` vraća vrednost `TRUE` ako je pronađena sledeća instanca parametra zadata vrednošću `paramCode` i njegova vrednost dodeljena promenljivoj `param`. Ako je pretraga bila ne uspešna funkcija vraća vrednost `FALSE`.

Parametri:

paramCode - Predstavlja kod parametra koji se traži u primljenoj poruci.

param - Promenljiva u koju će se smestiti vrednost traženog parametra veličine četiri bajta.

Povratna Vrednost:

Funkcija vraća TRUE ako je traženi parametar pronađen u primljenoj poruci i njegova vrednost dodeljena promenljivoj *param* u suprotnom funkcija vraća vrednost FALSE.

Napomena:

Prva instanca svakog parametra se mora preuzeti odgovarajućom funkcijom (koja u svom nazivu nema Next) za ovu funkciju to je [GetParamDWord\(\)](#). Upotreba next funkcija smanjuje vreme pretrage za parametrima unutar primljene poruke.

Nazad na [API funkcije](#)

GetNextParamWord

```
bool GetNextParamWord(    uint16 paramCode,  
                          WORD    &param );
```

Funkcija `GetNextParamWord()` vraća vrednost TRUE ako je pronađena sledeća instanca parametra zadata vrednošću *paramCode* i njegova vrednost dodeljena promenljivoj *param*. Ako je pretraga bila ne uspešna funkcija vraća vrednost FALSE.

Parametri:

paramCode - Predstavlja kod parametra koji se traži u primljenoj poruci.

param - Promenljiva u koju će se smestiti vrednost traženog parametra veličine dva bajta.

Povratna Vrednost:

Funkcija vraća TRUE ako je traženi parametar pronađen u primljenoj poruci i njegova vrednost dodeljena promenljivoj *param* u suprotnom funkcija vraća vrednost FALSE.

Napomena:

Prva instanca svakog parametra se mora preuzeti odgovarajućom funkcijom (koja u svom nazivu nema Next) za ovu funkciju to je [GetParamWord\(\)](#). Upotreba next funkcija smanjuje vreme pretrage za parametrima unutar primljene poruke.

Nazad na [API funkcije](#)

GetObjectId

```
virtual uint32 GetObjectId()
```

Funkcija `GetObjectId()` vraća vrednost jedinstvenog identifikacionog broja za dati tip automata.

Povratna Vrednost:

Funkcija vraća vrednost jedinstvenog identifikacionog broja za dati tip automata.

Napomena:

Sistem automata ažurira ovu vrednost za svaki automat onog trenutku kada se on dodaje funkcijom [Add\(\)](#).

Nazad na [API funkcije](#)

GetParam

```
uint8 *GetParam( uint16 paramCode )
```

Funkcija GetParam() vraća adresu traženog parametra u primljenoj poruci ili vrednost NULL ako traženi parametar nije pronađen.

Parametri:

paramCode - Predstavlja kod parametra koji se traži u primljenoj poruci.

Povratna Vrednost:

Funkcija vraća adresu traženog parametra u primljenoj poruci ili vrednost NULL ako traženi parametar nije pronađen.

Napomena:

Vraćena adresa predstavlja početak parametra. Sam parametar je formatiran u zavisnosti od načina na koji je kodirana poruka. Za StandardMessage parametar se sastoji od dva bajta koji sadrže kod parametra, jedan bajt za dužinu informacionog dela poruke i informacioni deo.

Nazad na [API funkcije](#)

GetParamByte

```
bool GetParamByte( uint16 paramCode,  
                   BYTE &param )
```

Funkcija GetParamByte() vraća vrednost TRUE ako je pronađen parametar zadat vrednošću *paramCode* i njegova vrednost dodeljena promenljivoj *param*. Ako je pretraga bila ne uspešna funkcija vraća vrednost FALSE.

Parametri:

paramCode - Predstavlja kod parametra koji se traži u primljenoj poruci.

param - Promenljiva u koju će se smestiti vrednost traženog parametra veličine jednog bajta.

Povratna Vrednost:

Funkcija vraća TRUE ako je traženi parametar pronađen u primljenoj poruci i njegova vrednost dodeljena promenljivoj *param* u suprotnom funkcija vraća vrednost FALSE.

Napomena:

Prva instanca svakog parametra se mora preuzeti ovom funkcijom. Upotreba [GetNextParamByte\(\)](#) funkcija smanjuje vreme pretrage za narednim parametrima unutar primljene poruke.

Nazad na [API funkcije](#)

GetParamDWord

```
bool GetParamDWord( uint16 paramCode,  
                   DWORD &param )
```

Funkcija `GetParamDWord()` vraća vrednost `TRUE` ako je pronađen parametar zadat vrednošću *paramCode* i njegova vrednost dodeljena promenljivoj *param*. Ako je pretraga bila ne uspešna funkcija vraća vrednost `FALSE`.

Parametri:

paramCode - Predstavlja kod parametra koji se traži u primljenoj poruci.
param - Promenljiva u koju će se smestiti vrednost traženog parametra veličine četiri bajta.

Povratna Vrednost:

Funkcija vraća `TRUE` ako je traženi parametar pronađen u primljenoj poruci i njegova vrednost dodeljena promenljivoj *param* u suprotnom funkcija vraća vrednost `FALSE`.

Napomena:

Prva instanca svakog parametra se mora preuzeti ovom funkcijom. Upotreba [GetNextParamDWord\(\)](#) funkcija smanjuje vreme pretrage za narednim parametrima unutar primljene poruke.

Nazad na [API funkcije](#)

GetParamWord

```
bool GetParamWord( uint16 paramCode, WORD &param )
```

Funkcija `GetParamWord()` vraća vrednost `TRUE` ako je pronađen parametar zadat vrednošću *paramCode* i njegova vrednost dodeljena promenljivoj *param*. Ako je pretraga bila ne uspešna funkcija vraća vrednost `FALSE`.

Parametri:

paramCode - Predstavlja kod parametra koji se traži u primljenoj poruci.
param - Promenljiva u koju će se smestiti vrednost traženog parametra veličine dva bajta.

Povratna Vrednost:

Funkcija vraća `TRUE` ako je traženi parametar pronađen u primljenoj poruci i njegova vrednost dodeljena promenljivoj *param* u suprotnom funkcija vraća vrednost `FALSE`.

Napomena:

Prva instanca svakog parametra se mora preuzeti ovom funkcijom. Upotreba [GetNextParamWord\(\)](#) funkcija smanjuje vreme pretrage za narednim parametrima unutar primljene poruke.

Nazad na [API funkcije](#)

GetProcedure

```
PROC_FUN_PTR GetProcedure( uint16 event )
```

Funkcija `GetProcedure()` vraća adresu funkcije koja je definisana da obradi događaj dat parametrom *event* u datom stanju automata.

Parametri:

event - Vrednost ovog parametra je događaj koji predstavlja kod primljene poruke.

Povratna Vrednost:

Funkcija vraća adresu funkcije koja je definisana za dati kod poruke i stanje u kojem se automat trenutno nalazi.

Napomena:

Struktura podataka u kojoj su smeštene sve informacije o stanjima, porukama koje automat očekuje i funkcijama za obradu primljenih poruka inicijalizuje se prilikom dodavanja svakog automata u sistem automata. Ove vrednosti se definišu u okviru metode [Initialize\(\)](#). Ako nije definisana funkcija za dato stanje i primljenu poruku funkcija će vratiti adresu funkcije [DoNothing\(\)](#) koja je predefinisana za neočekivane poruke.

Nazad na [API funkcije](#)

GetRightMbx

```
virtual inline uint8 GetRightMbx()
```

Funkcija `GetRightMbx()` vraća broj koji identifikuje poštansko sanduče automata koji se na logičkom nivou nalazi desno od automata koji je pozvao ovu funkciju.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje poštansko sanduče automata koji se logički posmatrano nalazi desno od trenutnog automata.

Nazad na [API funkcije](#)

GetRightAutomate

```
virtual inline uint8 GetRightAutomate()
```

Funkcija `GetRightAutomate()` vraća broj koji identifikuje tip automata koji se na logičkom nivou nalazi desno od automata koji je pozvao ovu funkciju.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje tip automata koji se logički posmatrano nalazi desno od trenutnog automata.

Nazad na [API funkcije](#)

GetRightGroup

```
virtual inline uint8 GetRightGroup()
```

Funkcija `GetRightGroup()` vraća broj koji identifikuje grupu kojoj pripadaju automati koji se na logičkom nivou nalaze desno od automata koji je pozvao ovu funkciju.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje grupu automata koji se logički posmatrano nalazi desno od trenutnog automata.

Nazad na [API funkcije](#)

GetRightObjectId

```
virtual inline uint32 GetRightObjectId();
```

Funkcija `GetRightObjectId()` vraća broj koji identifikuje automat koji se na logičkom nivou nalazi desno od automata koji je pozvao ovu funkciju.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje automat koji se logički posmatrano nalazi desno od trenutnog automata.

Nazad na [API funkcije](#)

GetState

```
virtual inline uint8 GetState()
```

Funkcija `GetState()` vraća broj koji identifikuje trenutno stanje automata.

Povratna Vrednost:

Funkcija vraća broj koji identifikuje trenutno stanje automata.

Nazad na [API funkcije](#)

IsBufferSmall

```
virtual bool IsBufferSmall(      uint8 *buff,  
                               uint32 length )
```

Funkcija `IsBufferSmall()` vraća vrednost `TRUE` ako je vrednost parametra *length* veća nego što je veličina bafera čija je adresa smeštena u parametru *buff* u suprotnom funkcija vraća vrednost `FALSE`.

Parametri:

buff - Parametar koji sadrži adresu bafera čija se veličina testira.
length - Vrednost sa kojom se poredi veličina datog bafera.

Povratna Vrednost:

Funkcija vraća vrednost `TRUE` ako je veličina bafera manja od vrednosti parametra *length* u suprotnom se vraća vrednost `FALSE`.

Nazad na [API funkcije](#)

Initialize

```
virtual void Initialize() = 0
```

Initialize() je prava virtuelna funkcija koja mora da se implementira za svaku klasu automata. U okviru ove funkcije se definišu funkcije za obradu primljenih poruka i vremenske kontrole koje koriste automati date klase.

Napomena:

U okviru funkcije Initialize() se definišu funkcije za obradu primljenih poruka ([InitEventProc\(\)](#), [InitUnexpectedEventProc\(\)](#)) i funkcije kojim se inicijalizaciju vremenske kontrole ([InitTimerBlock\(\)](#)).

Nazad na [API funkcije](#)

InitEventProc

```
void InitEventProc( uint8 state,
                   uint16 event,
                   PROC_FUN_PTR fun )
```

Funkcijom InitEventProc() se formira uređena trojka kojom se definiše koja funkcija će biti pozvana da obradi primljenu poruku u zavisnosti od stanja i koda primljene poruke.

Parametri:

state - Vrednost ovog parametra definiše stanje automata.
event - Vrednost ovog parametra definiše kod primljene poruke (dogadjaj).
fun - Adresa funkcije koja treba da se izvrši.

Napomena:

Ova funkcija se koristi isključivo u okviru metode [Initialize\(\)](#). Nizom poziva ove funkcije sa različitim parametrima se popunjava odgovarajuća struktura podataka na osnovu koje sistem automata u sprezi sa odgovarajućim automatom poziva funkcije za obradu događaja u zavisnosti od koda poruke i stanja automata.

Nazad na [API funkcije](#)

InitTimerBlock

```
void InitTimerBlock(      uint16 tmrId,
                          uint32 count,
                          uint16 signalId );
```

Funkcijom InitTimerBlock() se inicijalizuju elementi kernela za rukovanje vremenskim kontrolama. Sve vremenske kontrole koje će koristiti data klasa automata moraju se registrovati ovom funkcijom.

Parametri:

tmrId - Parametar čija vrednost identifikuje vremensku kontrolu.

count - Definiše broj vremenskih jedinica date vremenske kontrole.

signalId - Kod poruke koja će se poslati automatski kada istekne vremenska kontrola.

Napomena:

Identifikator se koristi za rukovanje vremenskom kontrolom, preko ove vrednosti se vremenska kontrola pokreće i zaustavlja. Vrednost parametra *count* predstavlja vreme potrebno da istekne data vremenska kontrola i nakon koje će automatu biti poslata poruka sa kodom koji je jednak vrednosti parametra *signalId*.

Nazad na [API funkcije](#)

InitUnexpectedEventProc

```
void InitUnexpectedEventProc(    uint8 state,  
                                PROC_FUN_PTR fun )
```

Funkcijom `InitUnexpectedEventProc()` se definiše funkcija koja će izvršavati obradu ne očekivanih poruka za dati automat ako se on nalazi u odgovarajućem stanju.

Parametri:

state - Parametar definiše stanje automata u kojem će se pozvati funkcija zadata parametrom *fun*.

fun - Adresa funkcije koja se definiše za obradu ne očekivanih poruka.

Napomena:

Ako se ne definiše odgovarajuća funkcija za obradu ne očekivanih poruka pozvaće se predefinisana funkcija [DoNothing\(\)](#).

Nazad na [API funkcije](#)

IsTimerRunning

```
bool IsTimerRunning( uint16 id )
```

Funkcija `IsTimerRunning()` se koristi za testiranje da li je data vremenska kontrola pokrenuta.

Parametri:

id - Predstavlja identifikator vremenske kontrole za koju se proverava da li je pokrenuta.

Povratna Vrednost:

Vrednost `TRUE` označava da je vremenska kontrola data sa identifikatorom *id* pokrenuta dok vrednost `FALSE` označava da data vremenska kontrola nije pokrenuta.

Nazad na [API funkcije](#)

NoFreeObjectProcedure

```
void NoFreeObjectProcedure( uint8 *msg )
```


Funkcija `NoFreeObjectProcedure()` definiše ponašanje automata u slučaju kada se koristi lista slobodnih automata i kada je ova lista prazna (svi automati su zauzeti).

Parametri:

`msg` - Adresa bafera u kojem je smeštena primljena poruka.

Napomena:

U okviru ove funkcije je implementirana procedura koju će sistem automata sprovesti u slučaju da su iscrpljeni resursi (nema slobodnih automata datog tipa). Dodatna obrada ovakvog događaja se implementira u okviru virtuelne funkcije `NoFreeInstances()`.

Nazad na [API funkcije](#)

NoFreeInstances

```
virtual void NoFreeInstances() = 0
```

Funkcija `NoFreeInstances()` definiše ponašanje sistema automata u slučaju kada se koristi lista slobodnih automata i kada je ova lista prazna (svi automati su zauzeti).

Napomena:

Ova funkcija je predviđena da implementira akcije koje sistem automata treba da preduzme ako nema slobodnih automata datog tipa.

Nazad na [API funkcije](#)

ParseMessage

```
virtual bool ParseMessage( uint8 *msg )
```

Funkcija `ParseMessage()` proverava da li je primljena poruka, koja se nalazu u baferu a koji je dat preko parametra `msg`, kodirana na odgovarajući način i dodeljuje adresu bafera u kojem se nalazi primljena poruka promenljivoj `CurrentMessage` (ovo je interna promenljiva koja se koristi za rukovanje primljenim porukama u okviru automata).

Parametri:

`msg` - Ovaj parametar sadrži adresu bafera u kojem je smeštena primljena poruka.

Povratna Vrednost:

Ako nije bilo grešaka prilikom provere primljene poruke za tip kodiranja i ako je adresa bafera sa primljenom porukom uspešno dodeljena promenljivoj `CurrentMessage` funkcija će vratiti vrednost `TRUE` u suprotnom došlo je do neke greške i vraćena vrednost će biti `FALSE`.

Napomena:

Sa svaku primljenu poruku ova funkcija se interno automatski poziva. Ako se u nekom baferu nalazi poruka koja je pravilno formatirana pre poziva bilo koje funkcije koje rukuju sa primljenom porukom mora se pozvati ova funkcija, stim da vrednost parametra `msg` mora sadržati vrednost tog bafera.

Nazad na [API funkcije](#)

PrepareNewMessage

```
virtual void PrepareNewMessage( uint8 *msg )
```

Funkcija `PrepareNewMessage()` dodeljuje adresu bafera datog parametrom *msg*, internoj promenljivoj `NewMessage` koja će se koristiti prilikom formiranja nove poruke.

Parametri:

msg - Ovaj parametar sadrži adresu bafera u kojem će biti smeštena nova poruka.

Napomena:

Pre nego što automat pošalje novu poruku mora se inicijalizovati interna promenljiva `NewMessage`. Tek nakom što je izvedena inicijalizacija sa odgovarajućom funkcijom `PrepareNewMessage()` mogu se koristiti ostale funkcije koje rade sa novom porukom.

Nazad na [API funkcije](#)

PrepareNewMessage

```
virtual void PrepareNewMessage(  uint32 length,  
                                uint16 code,  
                                uint8 infoCode=LOCAL_PARAM_CODING )
```

Funkcija `PrepareNewMessage()` se koristi pri formiranju nove poruke. Funkcija uzima bafer od kernela pri čemu je veličina bafera veća ili jednaka sa parametrom *length*. Kod poruke i tip kodiranja informacionog dela poruke ako je prosleđen funkciji se upisuju u zaglavlje nove poruke automatski.

Parametri:

length - Parametar čija vrednost predstavlja minimalnu veličinu bafera za novu poruku.

code - Vrednost ovog parametra predstavlja kod nove poruke.

infoCode - Tip kodiranja koji će se koristiti za informacioni deo poruke.

Napomena:

Vrednost parametra *length* može imati vrednost nula pošto najmanji bafer u sistemu automata po pravilu ima dovoljno mesta da se u njega smesti zaglavlje poruke ali ako je veličina bafera nije dovoljna za parametre koji se naknadno dodaju u poruku doći će do vraćanja starog bafera i uzimanja novog većeg u koji može da se smesti nova poruka. Kako dodatne operacije zahtevaju vreme u slučaju vremenski kritičnih događaja poželjno je veličinu bafera odmah dobro odrediti.

Nazad na [API funkcije](#)

Process

```
virtual void Process( uint8 *msg )
```

Funkcija `Process()` izvršava pripremu za obradu primljene poruke i odabira funkciju za obradu primljenu poruke u zavisnosti od koda poruke i stanja automata. Nako što se obradi primljena poruka funkcija vraća zauzete resurse sistemu automata.

Parametri:

`msg` - Ovaj parametar sadrži adresu bafera u kojem se nalazi primljena poruka.

Napomena:

Ova funkcija se interno poziva od strane sistema automata. Pošto je ova funkcija virtuelna moguće je po potrebi redefinisati način na koji se obrađuju primljene poruke.

Nazad na [API funkcije](#)

PurgeMailBox

```
void PurgeMailBox()
```

Funkcija `PurgeMailBox()` izbacuje sve primljene poruke iz poštanskog sandučeta u koji pristižu nove poruke za dati tip automata. Baferi svih obrisanih poruka se vraćaju nazad u sistem automata.

Napomena:

Poštansko sanduče se identifikuje preko automata iz kog se poziva ova funkcija ali ova funkcija nije vezana za instancu automata, SVE pristigle poruke u datom poštanskom sandučetu biće obrisane.

Nazad na [API funkcije](#)

RemoveParam

```
bool RemoveParam( uint16 paramCode )
```

Funkcija `RemoveParam()` uklanja parametar iz nove poruke. Funkcija pretražuje novu poruku za datu vrednost koda traženog parametra i uklanja odgovarajući parametar iz poruke.

Parametri:

`paramCode` - Ova vrednost je kod parametra koji treba da se ukloni iz nove poruke.

Povratna Vrednost:

Ako je dati parametar uspešno pronađen i uklonjen iz nove poruke funkcija vraća vrednost `TRUE` u suprotnom funkcija vraća vrednost `FALSE`.

Napomena:

Uklanjanje parametra sa vrednošću 0 nije poželjno pošto on predstavlja kraj parametara u poruci. U debug verziji biće prijavljena greška i program će prestati sa radom ako se ukloni ovaj parametar.

Nazad na [API funkcije](#)

Reset

```
virtual void Reset()
```

Funkcija `Reset()` dovodi automat u njegovo inicijalno stanje i zaustavlja sve vremenske kontrole koje su u okviru njega pokrenute.

Napomena:

Ukoliko je potrebno da se izvrše dodatne akcije prilikom dovodenja automata u njegovo inicijalno stanje ovu funkciju je potrebno redefinisati u okviru klase koja implementira dati automat.

Nazad na [API funkcije](#)

ResetTimer

```
void ResetTimer( uint16 id )
```

Funkcija `ResetTimer()` se koristi nako što je istekla data vremenska kontrola da bi se u internoj strukturi podataka sistema automata označilo da je vraćen bafer koji je ta vremenska kontrola uzela kada je startovana.

Parametri:

`id` - Identifikator vremenske kontrole za koju treba oznaciti da joj je vraćen bafer.

Nazad na [API funkcije](#)

RestartTimer

```
void RestartTimer( uint16 tmrId )
```

Funkcija `RestartTimer()` se koristi kad je potrebno da se vremenska kontrola zaustavi i ponovo pokrene.

Parametri:

`tmrId` - Identifikator vremenske kontrole koju treba prvo zaustaviti i zatim ponovo pokrenuti.

Nazad na [API funkcije](#)

RetBuffer

```
virtual void RetBuffer( uint8 *buff );
```

Funkcija `RetBuffer` se koristi za vraćanje bafera koji su od kernela preuzeti funkcijom [GetBuffer\(\)](#).

Parametri:

`buff` - Pokazivač na bafer koji se vraća nazad u kernel.

Napomena:

Svi baferi koji se uzimaju od kernela moraju se vratiti kada više nisu potrebni kako bi se izbegao slučaj kada nema više slobodnih bafera što izazva generisanje izuzetka koji ako se ne obradi dovodi do prekida rada programa.

Nazad na [API funkcije](#)

ReturnMsg

```
void ReturnMsg( uint8 mbxId )
```

Funkcija ReturnMsg() generise novu poruku koja predstavlja kopiju primljene poruke i šalje je u poštansko sanduče koje je definisano parametrom funkcije.

Parametri:

mbxId - Identifikacioni broj poštanskog sandučeta u koji šaljemo kopiju primljene poruke.

Nazad na [API funkcije](#)

SetBitParamByteBasic

```
void SetBitParamByteBasic( BYTE param,
                           uint32 offset,
                           uint32 mask = MASK_32_BIT )
```

Funkcija SetBitParamByteBasic() se koristi da u novoj poruci na mestu definisanom sa parametrom *offset* izvrši: binarnu operaciju AND između stare vrednosti na datoj memorijskoj lokaciji i vrednosti parametra *mask* a zatim da sa tako dobijenom vrednošću da izvrši binarnu operaciju OR sa vrednošću parametra *param*.

Parametri:

param - Parametar veličine jednog bajta.

offset - Definiše mesto od početka poruke gde će biti izvršene pomenute binarne operacije.

mask - Parametar kojim se maskira stara vrednost memorijske lokacije.

Nazad na [API funkcije](#)

SetBitParamDWordBasic

```
void SetBitParamDWordBasic(   DWORD param,
                              uint32 offset,
                              uint32 mask = MASK_32_BIT );
```

Funkcija SetBitParamDWordBasic() se koristi da u novoj poruci na mestu definisanom sa parametrom *offset* izvrši: binarnu operaciju AND između stare vrednosti na datoj memorijskoj lokaciji i vrednosti parametra *mask* a zatim da sa tako dobijenom vrednošću da izvrši binarnu operaciju OR sa vrednošću parametra *param*.

Parametri:

param - Parametar veličine četiri bajta.

offset - Definiše mesto od početka poruke gde će biti izvršene pomenute binarne operacije.

mask - Parametar kojim se maskira stara vrednost memorijske lokacije.

Nazad na [API funkcije](#)

SetBitParamWordBasic

```
void SetBitParamWord(      WORD param,
                          uint32 offset,
                          uint32 mask = MASK_32_BIT );
```

Funkcija `SetBitParamWordBasic()` se koristi da u novoj poruci na mestu definisanom sa parametrom *offset* izvrši binarnu operaciju AND između stare vrednosti na datoj memorijskoj lokaciji i vrednosti parametra *mask* a zatim da sa tako dobijenom vrednošću da izvrši binarnu operaciju OR sa vrednošću parametra *param*.

Parametri:

param - Parametar veličine dva bajta.

offset - Definiše mesto od početka poruke gde će biti izvršene pomenute binarne operacije.

mask - Parametar kojim se maskira stara vrednost memorijske lokacije.

Nazad na [API funkcije](#)

SetCallId

```
inline void SetCallId()
```

Funkcija `SetCallId()` dodeljuje vrednost zaštićenom atributu koji predstavlja jedinstveni identifikator trenutnog poziva. Ova funkcija automatski dodeljuje nove vrednosti.

Nazad na [API funkcije](#)

SetCallId

```
inline void SetCallId( uint32 id )
```

Funkcija `SetCallId()` dodeljuje vrednost zaštićenom atributu koji predstavlja jedinstveni identifikator trenutnog poziva.

Parametri:

id – Nova vrednost atributa u kojem se čuva vrednost identifikatora trenutnog poziva.

Napomena:

Ako se koristi ova funkcija i atribut za praćenje trenutnih poziva potrebno je obezbediti generisanje identifikatora tako da se ne desi slučaj da dva različita poziva imaju iste identifikatore.

Nazad na [API funkcije](#)

SetCallIdFromMsg

```
inline void SetCallIdFromMsg()
```

Funkcija `SetCallIdFromMsg()` se koristi da atributu za praćenje trenutnog poziva dodeli vrednost koja je dobijena u primljenoj poruci.

Nazad na [API funkcije](#)

SetDefaultFSMData

```
virtual void SetDefaultFSMData() = 0
```

Funkcija SetDefaultFSMData() je prava virtuelna funkcija koja služi da se u njoj definiše inicijalizacija atributa koje dati automat treba da postavi na odgovarajuće vrednosti pre početka svog rada.

Napomena:

Automati izvedeni iz klase [FiniteStateMachine](#) moraju implementirati ovu funkciju.

Nazad na [API funkcije](#)

SetDefaultHeader

```
virtual void SetDefaultHeader( uint8 infoCoding ) = 0
```

Funkcija SetDefaultHeader() je prava virtuelna funkcija u kojoj se definiše automatsko popunjavanje zaglavlja svake poruke, najčešće se radi o podacima automata koji šalje poruku.

Parametri:

infoCoding - Definiše različita zaglavlja u zavisnosti od tipa kodiranja poruke.

Napomena:

Automati izvedeni iz klase [FiniteStateMachine](#) moraju implementirati ovu funkciju..

Nazad na [API funkcije](#)

SetGroup

```
inline void SetGroup( uint8 id )
```

Funkcija SetGroup() se koristi da automatu dodeli novu vrednost atributu koji identifikuje grupu kojoj dati automat pripada.

Parametri:

id - Parametar kojim se identifikuje grupa u kojoj pripada dati automat.

Nazad na [API funkcije](#)

SetInitialState

```
virtual void SetInitialState()
```

Funkcija SetInitialState() postavlja stanje automata na inicijalno stanje.

Napomena:

Inicijalno stanje automata je ono stanje automata za koje je definisano da ima vrednost 0.

Nazad na [API funkcije](#)

SetKernelObjects

```
static void SetKernelObjects(    TPostOffice *postOffice,  
                                TBuffers *buffers,  
                                CTimer *timer )
```

Funkcija SetKernelObjects() se koristi da kernelu pridruži sistemske objekte koji su jedinstveni za sve automate. SetKernelObjects ima tri parametra i to su pokazivači na objekat za rukovanje porukama, objekat za rukovanje memorijom i objekat za rukovanje vremenskim kontrolama.

Parametri:

postOffice - Pokativač na objekat za rukovanje porukama.
buffers - Pokazivač na objekat za rukovanje memorijom.
timer - Pokazivač na objekat za rukovanje vremenskim kontrolama.

Napomena:

Ova funkcija se poziva za potrebe sistema automata u okviru funkcije [InitKernel](#). Kako su parametri ove funkcije objekti koji su jedinstveni za sve automate ova funkcija mora da se poziva sa posebnom pažnjom jer promena ovih parametara u toku rada sistema automata može da dovede do ne definisanog rada sistema automata.

Nazad na [API funkcije](#)

SetLeftMbx

```
inline void SetLeftMbx( uint8 mbx )
```

Funkcija SetLeftMbx() postavlja vrednost atributa za poštansko sanduče logički levog automata u odnosu na dati automat.

Parametri:

mbx - Identifikator poštanskog sandučeta logički levog automata u odnosu na dati automat.

Nazad na [API funkcije](#)

SetLeftAutomate

```
inline void SetLeftAutomate( uint8 automate )
```

Funkcija SetLeftAutomate() postavlja vrednost atributa koji definiše tip levog automata u odnosu na dati automat.

Parametri:

automate - Tip levog automata.

Nazad na [API funkcije](#)

SetLeftObject

```
inline void SetLeftObject( uint8 group )
```

Funkcija SetLeftObject() postavlja vrednost atributa koji predstavlja grupu kojoj pripada levi automat u odnosu na dati automat.

Parametri:

group - Identifikator grupe kojoj pripada levi automat.

Nazad na [API funkcije](#)

SetLeftObjectId

```
inline void SetLeftObjectId( uint32 id )
```

Funkcija SetLeftObjectId() postavlja vrednost atributa za jedinstveni identifikator logički levog automata u odnosu na dati automat.

Parametri:

id - Jedinstveni identifikator levog automata.

Nazad na [API funkcije](#)

SetLogInterface

```
static void SetLogInterface( LogInterface *loggingObject )
```

Funkcijom SetLogInterface() se definiše koji će objekat biti korišćen za vođenje evidencije o toku rada rada sistema automata. U zavisnosti od samog objekta ova evidencija može da se snima u lokalnu datoteku ili u neku datoteku preko mreže. Dobijena datoteka se koristi za otkrivanje grešaka u toku testiranja sistema automata ili u toku normalnog rada sistema automata za praćenje ispravnog rada celog sistema.

Parametri:

loggingObject - Pokazivač na objekat koji će vršiti evidenciju rada sistema automata.

Napomena:

Inicijalizacija objekta za vođenje evidencije mora biti posle inicijalizacije automata.

Nazad na [API funkcije](#)

SendMessage

```
inline void SendMessage( uint8 mbxId )
```

Funkcija SendMessage() se koristi za slanje nove poruke u željeno poštansko sanduče.

Parametri:

`mbxId` - Parametar kojim se definiše poštansko sanduče u koje želimo da pošaljemo novu poruku.

Napomena:

Nova poruka je poruka na koju pokazuje interni pokazivač `NewMessage` a koji je inicijalizovan funkcijom [PrepareNewMessage\(\)](#).

Nazad na [API funkcije](#)

SendMessage

```
inline void SendMessage(  uint8 mbxId,
                          uint8 *msg );
```

Funkcija `SendMessage()` se koristi za slanje poruke definisane parametrom `msg` u željeno poštansko sanduče.

Parametri:

`mbxId` - Parametar kojim se definiše poštansko sanduče u koje želimo da pošaljemo poruku.

`msg` - Pokazivač na bafer u kojem se nalazi pravilno formatirana poruka.

Nazad na [API funkcije](#)

SetMessageFromData

```
void SetMessageFromData()
```

Funkcijom `SetMessageFromData()` se u zaglavlje nove poruke postavljaju podaci o automatu koji šalje poruku. Podaci koji se postavljaju u zaglavlje poruke su tip automata, grupa i identifikacioni broj automata koji šalje poruku.

Napomena:

Ova funkcija se automatski poziva u okviru funkcije [SendMessage\(\)](#).

Nazad na [API funkcije](#)

SetMsgCallId

```
inline void SetMsgCallId( uint32 id )
```

Funkcija `SetMsgCallId()` se koristi da postavi jedinstveni identifikacioni broj za atribut koji identifikuje jedan poziv.

Parametri:

`id` - Jedinstveni identifikacioni broj poziva.

Napomena:

Ovaj parametar je isti za sve poruke koje su vezane za obradu istog poziva.

Nazad na [API funkcije](#)

SetMsgCallId

```
inline void SetMsgCallId( uint32 id,
                          uint8 *msg )
```

Funkcija SetMsgCallId() se koristi da postavi jedinstveni identifikacioni broj za atribut koji identifikuje jedan poziv.

Parametri:

id - Jedinstveni identifikacioni broj poziva.
msg - Pokazivač na bafer u kojem se nalazi poruka.

Napomena:

Ovaj parametar je isti za sve poruke koje su vezane za obradu istog poziva.

Nazad na [API funkcije](#)

SetMsgCode

```
inline void SetMsgCode( uint16 code )
```

Funkcija SetMsgCode() se koristi za postavljanje koda poruke u novu poruku.

Parametri:

code - Parametar koji definiše kod nove poruke.

Nazad na [API funkcije](#)

SetMsgCode

```
inline void SetMsgCode( uint16 code,
                          uint8 *msg );
```

Funkcija SetMsgCode() se koristi za postavljanje koda poruke u poruku koja se nalazi u baferu koji je dat parametrom *msg*.

Parametri:

code - Parametar koji definiše kod nove poruke.
msg - Pokazivač na bafer u kojem se nalazi poruka kojoj treba da se postavi kod poruke.

Nazad na [API funkcije](#)

SetMsgFromAutomate

```
inline void SetMsgFromAutomate( uint8 from )
```

Funkcija SetMsgFromAutomate() se koristi da u novoj poruci definiše automat koji je poslao poruku.

Parametri:

from - Parametar koji identifikuje automat koji je poslao poruku.

Napomena:

Ova funkcija se automatski poziva preko funkcije [SetMessageFromData\(\)](#).

Nazad na [API funkcije](#)

SetMsgFromAutomate

```
inline void SetMsgFromAutomate(  uint8 from,
                                uint8 *msg );
```

Funkcija `SetMsgFromAutomate()` se koristi da u poruci koja se nalazi u baferu datom sa parametrom *msg* definiše automat koji je poslao poruku.

Parametri:

from - Parametar koji identifikuje automat koji je poslao poruku.

msg - Pokazivač na bafer u kojem se nalazi poruka.

Nazad na [API funkcije](#)

SetMsgFromGroup

```
inline void SetMsgFromGroup( uint8 from )
```

Funkcija `SetMsgFromGroup()` se koristi da u novoj poruci definiše grupu kojoj pripada automat koji je poslao poruku.

Parametri:

from - Parametar koji identifikuje grupu kojoj pripada automat koji je poslao poruku.

Napomena:

Ova funkcija se automatski poziva preko funkcije [SetMessageFromData\(\)](#).

Nazad na [API funkcije](#)

SetMsgFromGroup

```
inline void SetMsgFromGroup(  uint8 from,
                                uint8 *msg )
```

Funkcija `SetMsgFromGroup()` se koristi da u poruci koja se nalazi u baferu datom sa parametrom *msg* definiše grupu kojoj pripada automat koji je poslao poruku.

Parametri:

from - Parametar koji identifikuje grupu kojoj pripada automat koji je poslao poruku.

msg - Pokazivač na bafer u kojem se nalazi poruka.

Nazad na [API funkcije](#)

SetMsgInfoCoding

```
inline void SetMsgInfoCoding( uint8 codingType )
```

Funkcija SetMsgInfoCoding() postavlja u zaglavlju nove poruke vrednost parametra kojom se definiše tip kodiranja poruke.

Parametri:

`codingType` - Parameter kojim se definiše tip kodiranja nove poruke.

Napomena:

Ova funkcija se automatski poziva u okviru funkcije [PrepareNewMessage\(\)](#).

Nazad na [API funkcije](#)

SetMsgInfoCoding

```
inline void SetMsgInfoCoding(      uint8 codingType,  
                                uint8 *msg )
```

Funkcija SetMsgInfoCoding() postavlja u zaglavlju poruke, koja se nalazi u baferu datom sa parametrom *msg*, vrednost kojom se definiše tip kodiranja poruke.

Parametri:

`codingType` - Tip kodiranja nove poruke.

`msg` - Pokazivač na bafer u kojem se nalazi poruka kojoj treba da se postavi tip kodiranja poruke.

Nazad na [API funkcije](#)

SetMsgInfoLength

```
inline void SetMsgInfoLength( uint16 length )
```

Funkcijom SetMsgInfoLength() se u zaglavlju svake poruke postavlja vrednost za dužinu informacionog dela poruke (zaglavlje je isključeno).

Parametri:

`length` - Parametar čija vrednost predstavlja dužinu informacionog dela poruke.

Napomena:

Sve funkcije kernela koje dodaju parametre u novu poruku automatski pozivaju ovu funkciju i ažuriraju vrednost u zaglavlju poruke koja predstavlja dužinu informacionog dela poruke.

Nazad na [API funkcije](#)

SetMsgInfoLength

```
inline void SetMsgInfoLength(    uint16 length,  
                                uint8 *msg );
```

Funkcijom SetMsgInfoLength() se u zaglavlju poruke koja se nalazi u baferu koji je dat parametrom *msg* postavlja vrednost za dužinu informacionog dela poruke (zaglavlje je isključeno).

Parametri:

length - Parametar čija vrednost predstavlja dužinu informacionog dela poruke.
msg - Pokazivač na bafer u kojem se nalazi poruka.

Nazad na [API funkcije](#)

SetMsgObjectNumberFrom

```
inline void SetMsgObjectNumberFrom( uint32 from )
```

Funkcija SetMsgObjectNumberFrom() se koristi da u novoj poruci postavi jedinstveni identifikator automata koji je poslao poruku.

Parametri:

from - Parametar koji predstavlja jedinstveni identifikator automata koji je poslao poruku.

Napomena:

Ova funkcija se automatski poziva preko funkcije [SetMessageFromData\(\)](#).

Nazad na [API funkcije](#)

SetMsgObjectNumberFrom

```
inline void SetMsgObjectNumberFrom(    uint32 from,  
                                uint8 *msg )
```

Funkcija SetMsgObjectNumberFrom() se koristi da u poruci koja se nalazi u baferu koji je dat sa parametrom *msg* postavi jedinstveni identifikator automata koji je poslao poruku.

Parametri:

from - Parametar koji predstavlja jedinstveni identifikator automata koji je poslao poruku.
msg - Pokazivač na bafer u kojem se nalazi poruka.

Nazad na [API funkcije](#)

SetMsgObjectNumberTo

```
inline void SetMsgObjectNumberTo( uint32 to )
```

Funkcija SetMsgObjectNumberTo() se koristi da u zaglavlju nove poruke postavi identifikator automata kojem se šalje poruka.

Parametri:

to - Parametar čija je vrednost jedinstveni identifikator automata kojem se šalje nova poruka.

Nazad na [API funkcije](#)

SetMsgObjectNumberTo

```
inline void SetMsgObjectNumberTo( uint32 to, uint8 *msg )
```

Funkcija `SetMsgObjectNumberTo()` se koristi da u zaglavlju poruke koja se nalazi u baferu koji je dat parametrom *msg* postavi identifikator automata kojem se šalje poruka.

Parametri:

to - Parametar čija vrednost predstavlja jedinstveni identifikator automata kojem se šalje poruka.

msg - Pokazivač na bafer u kojem se nalazi poruka.

Nazad na [API funkcije](#)

SetMsgToAutomate

```
inline void SetMsgToAutomate( uint8 to )
```

Funkcija `SetMsgToAutomate()` se koristi da u zaglavlju nove poruke postavi tip automata kojem se šalje poruka.

Parametri:

to - Parametar čija vrednost predstavlja tip automata kojem se šalje poruka.

Nazad na [API funkcije](#)

SetMsgToAutomate

```
inline void SetMsgToAutomate(    uint8 to,  
                                uint8 *msg );
```

Funkcija `SetMsgToAutomate()` se koristi da u zaglavlju poruke koja se nalazi u baferu koji je dat parametrom *msg* postavi tip automata kojem se šalje poruka.

Parametri:

to - Parametar čija vrednost predstavlja tip automata kojem se šalje poruka.

msg - Pokazivač na bafer u kojem se nalazi poruka.

Nazad na [API funkcije](#)

SetMsgToGroup

```
inline void SetMsgToGroup( uint8 to )
```

Funkcija `SetMsgToGroup()` se koristi da u zaglavlju nove poruke postavi grupu kojoj pripada automat kojem se šalje poruka.

Parametri:

`to` - Parametar čija vrednost predstavlja grupu kojoj pripada automat kojem se šalje poruka.

Nazad na [API funkcije](#)

SetMsgToGroup

```
inline void SetMsgToGroup( uint8 to, uint8 *msg )
```

Funkcija `SetMsgToGroup()` se koristi da u zaglavlju poruke koja se nalazi u baferu koji je dat parametrom `msg` postavi grupu kojoj pripada automat kojem se šalje poruka.

Parametri:

`to` - Parametar čija vrednost predstavlja grupu kojoj pripada automat kojem se šalje poruka.

`msg` - Pokazivač na bafer u kojem se nalazi poruka.

Nazad na [API funkcije](#)

SendMessageLeft

```
void SendMessageLeft()
```

Funkcija `SendMessageLeft()` se koristi da pošalje novu poruku logički levom automatu.

Napomena:

Ova funkcija se koristi kada je moguće definisati logički levi automat. Tada je podešavanjem odgovarajućih parametara za levi automat moguće slati poruke ovom automatu samo pozivom ove funkcije koja automatski popunjava zaglavlje nove poruke potrebnim podacima o automatu koji šalje poruku i automatu koji prima poruku.

Nazad na [API funkcije](#)

SendMessageRight

```
void SendMessageRight()
```

Funkcija `SendMessageRight()` se koristi da pošalje novu poruku logički desnom automatu.

Napomena:

Ova funkcija se koristi kada je moguće definisati logički desni automat. Tada je podešavanjem odgovarajućih parametara za desni automat moguće slati poruke ovom automatu samo pozivom ove funkcije koja automatski popunjava zaglavlje nove poruke potrebnim podacima o automatu koji šalje poruku i automatu koji prima poruku.

Nazad na [API funkcije](#)

SetNewMessage

```
inline void SetNewMessage( uint8 *msg )
```

Funkcija SetNewMessage() se koristi kada treba da se internom atributu NewMessage koji pokazuje na novu poruku dodeli adresa bafera u kojem je već formatirana nova poruka ili u kojem treba da se formira nova poruka.

Parametri:

`msg` - Pokazivač na bafer u kojem će biti smeštena nova poruka.

Nazad na [API funkcije](#)

SetObjectId

```
inline void SetObjectId( uint32 id )
```

Funkcija SetObjectId() se koristi da dodeli jedinstveni identifikacioni broj automatu.

Parametri:

`id` - Jedinstveni identifikator automata.

Napomena:

Ova funkcija se automatski poziva prilikom dodavanja svakog automata funkcijom [Add\(\)](#) u sistem automata.

Nazad na [API funkcije](#)

SetRightMbx

```
inline void SetRightMbx( uint8 mbx )
```

Funkcija SetRightMbx() postavlja vrednost atributa za poštansko sanduče logički desnog automata u odnosu na dati automat.

Parametri:

`mbx` - Identifikator poštanskog sandučeta logički levog automata u odnosu na dati automat.

Nazad na [API funkcije](#)

SetRightAutomate

```
inline void SetRightAutomate( uint8 automate )
```

Funkcija SetRightAutomate() postavlja vrednost atributa koji definiše tip desnog automata u odnosu na dati automat.

Parametri:

`automate` - Tip desnog automata.

Nazad na [API funkcije](#)

SetRightObject

```
inline void SetRightObject( uint8 group )
```

Funkcija `SetRightObject()` postavlja vrednost atributa koji predstavlja grupu kojoj pripada desni automat u odnosu na dati automat.

Parametri:

`group` - Identifikator grupe kojoj pripada levi automat.

Nazad na [API funkcije](#)

SetRightObjectId

```
inline void SetRightObjectId( uint32 id )
```

Funkcija `SetRightObjectId()` postavlja vrednost atributa za jedinstveni identifikator logički desnog automata u odnosu na dati automat.

Parametri:

`id` - Jedinstveni identifikator desnog automata.

Nazad na [API funkcije](#)

SetState

```
inline void SetState( uint8 state )
```

Funkcija `SetState()` se koristi da postavi stanje automata u novo stanje definisano parametrom funkcije.

Parametri:

`state` - Parametar čija vrednost predstavlja novo stanje automata.

Nazad na [API funkcije](#)

StartTimer

```
void StartTimer( uint16 tmrId )
```

Funkcija `StartTimer()` se koristi za pokretanje vremenske kontrole čiji je identifikator dat parametrom *tmrId*.

Parametri:

`tmrId` - Identifikator vremenske kontrole koju treba pokrenuti.

Nazad na [API funkcije](#)

StopTimer

```
void StopTimer( uint16 tmrId )
```

Funkcija StopTimer() se koristi da zaustavi vremensku kontrolu čiji je identifikator dat parametrom *tmrId*.

Parametri:

tmrId - Identifikator vremenske kontrole koju treba zaustaviti.

Nazad na [API funkcije](#)

SysClearLogFlag

```
static void SysClearLogFlag()
```

Funkcija SysClearLogFlag() se koristi kada treba da se zaustavi vođenje evidencije poruka koje razmenjuju automati u sastavu sistema automata.

Nazad na [API funkcije](#)

SysStartAll

```
static void SysStartAll()
```

Funkcija SysStartAll() se koristi kada treba da se pokrene vođenje evidencije poruka koje razmenjuju automati u sastavu sistema automata.

Napomena:

Inicijalno stanje sistema automata podrazumeva da je evidencija aktivirana.

Nazad na [API funkcije](#)

NetFSM

```
NetFSM(      uint16 numOfTimers = DEFAULT_TIMER_NO,  
             uint16 numOfState  = DEFAULT_STATE_NO,  
             uint16 maxNumOfProceduresPerState = DEFAULT_PROCEDURE_NO_PER_STATE,  
             bool getMemory = true );
```

Konstruktor NetFSM() ima osnovni zadatak da napravi instancu datog automata i da inicijalizuje strukture podataka koje su potrebne za njegov pravilan rad.

Parametri:

numOfTimers - Broj vremenskih kontrola koje će koristiti ovaj automat.

numOfState - Broj stanja koje imaju automati ovog tipa.

maxNumOfProceduresPerState - Maksimalan broj funkcija prelaza za jedno stanje automata.

getMemory - Indikator na osnovu kog konstruktor rezerviše memoriju za stanja i funkcije prelaza

Napomena:

Konstruktor je moguće pozvati bez parametara, tada će se automat inicijalizovati sa predefinisanim vrednostima. Parametar getMemory se koristi kada je potrebno izvršiti optimizaciju zauzimanja slobodne memorije za tabelu sa funkcijama prelaza.

Nazad na [API funkcije](#)

convertFSMToNetMessage

```
virtual void convertFSMToNetMessage() = 0
```

Funkcija convertFSMToNetMessage() se koristi da konvertuje format poruke koji automati razmenjuju u okviru sistema automata u format poruke koja se šalje preko TCP/IP protokola.

Napomena:

Ovo je prava virtuelna funkcija koja se mora implementirati u klasi koja nasleđuje klasu NetFSM.

Nazad na [API funkcije](#)

convertNetToFSMMessage

```
virtual uint16 convertNetToFSMMessage() = 0
```

Funkcija convertNetToFSMMessage() se koristi da konvertuje format poruke koji je primljen od strane TCP/IP protokola u format poruke koji automati razmenjuju u okviru sistema automata.

Povratna Vrednost:

Funkcija vraća kod primljen poruke.

Napomena:

Ovo je prava virtuelna funkcija koja se mora implementirati u klasi koja nasleđuje klasu NetFSM.

Nazad na [API funkcije](#)

establishConnection

```
void establishConnection()
```

Funkcija establishConnection() se koristi za uspostavljanje veze između dva sistema automata.

Napomena:

Preduslov za razmenu poruka pozivanjem funkcije [sendToTCP\(\)](#) mora biti uspostavljena veza između dva sistema automata.

Nazad na [API funkcije](#)

getProtocolInfoCoding

```
virtual uint8 getProtocolInfoCoding() = 0
```

Funkcija `getProtocolInfoCoding()` vraća vrednost tipa kodiranja poruke.

Povratna Vrednost:

Tip kodiranja poruke (tip kodiranja standardnih poruka ima vrednost 0).

Nazad na [API funkcije](#)

sendToTCP

```
void sendToTCP()
```

Funkcija `sendToTCP()` se koristi kada treba da se pošalje poruka iz jednog sistema automata u drugi sistem automata.

Nazad na [API funkcije](#)

9. Primer realizacije dva automata

U ovom dodatku biće prikazan primer implementacije automata i sistema automata. Automati pripadaju istoj klasi i njihov zadatak je nakon prve poruke da razmenjuju poruke dok se ne dostigne broj zadati broj poruka.

Datoteka Automat.h

```
#ifndef __AUTOMAT__
#define __AUTOMAT__

#include <stdio.h>
#include "stdlib.h"
#include "kernel\fsm.h"
#include "kernel\errorObject.h"
#include "Konstante.h"

class Automat: public FiniteStateMachine
{
    private:
        StandardMessage StandardMsgCoding;
        MessageInterface *GetMessageInterface( uint32 id );

        void SetDefaultHeader( uint8 infoCoding );
        uint8 GetMbxId();
        uint8 GetAutomate();
        void SetDefaultFSMData();
        void NoFreeInstances();

        uint8 text[20];
        uint32 msgNumber;
        uint32 idToMsg;

        // Fuunkcije prelaza Stanje IDLE
        void Automat_IDLE_START();
        void Automat_IDLE_MSG();
        // Fuunkcije prelaza Stanje MSG
        void Automat_MSG_MSG();
        void Automat_MSG_STOP();
        // Neocekivane poruke u stanjima IDLE i MSG
        void Automat_UNEXPECTED_IDLE();
        void Automat_UNEXPECTED_MSG();

    public:
        Automat();
        ~Automat(){};

        void Initialize();
        void StartDemo();
};
#endif
```

Datoteka Automat.cpp

```
#include "kernel/LogFile.h"
#include "Automat.h"

Automat::Automat() : FiniteStateMachine(
    0, //uint16 numOfTimers = DEFAULT_TIMER_NO,
    2, //uint16 numOfState = DEFAULT_STATE_NO,
    3 )//uint16 maxNumOfProceduresPerState = DEFAULT_PROCEDURE_NO_PER_STATE
{
    SetDefaultFSMData();
}

// Ova metoda sadrzi informacije o tome koje poruke automat moze da obradi
// Za ovakvu definiciju u okviru klase potrebno je definisati dodatnu
// promenljivu: StandardMessage standardMsgCoding;
MessageInterface *Automat::GetMessageInterface( uint32 id )
{
    switch( id )
    {
        case 0x00:
            return &StandardMsgCoding;
    }
    throw TErrorObject( __LINE__, __FILE__, 0x01010400);
}

// Ova metoda popunjava zaglavlje poruke koju automati medjusobno razmenjuju
void Automat::SetDefaultHeader( uint8 infoCoding )
{
    SetMsgInfoCoding( infoCoding );
    SetMessageFromData();
}

// Metoda kojom se definise broj postanskog sanduceta iz kog ce automati ove
// klase primati poruke koje su pristigle u sistem automata.
uint8 Automat::GetMbxId()
{
    return MBX_AUTOMAT_ID;
}

// Metoda vraca broj koji identifikuje tip automata kojem pripada ovaj automat
uint8 Automat::GetAutomate()
{
    return FSM_TYPE_AUTOMAT;
}

// Metoda u kojoj se izvrsava inicijalizacija atributa instanci ove klase.
void Automat::SetDefaultFSMData()
{
    msgNumber = 0;
    idToMsg = INVALID_ID_32;
}

// Metoda koja ce se pozvati ako nema više slobodnih automata da obrade
// pristiglu poruku. Ova metoda se koristi ako su automati ove klase predhodno
// dodati u sistem automata sa parametrom useFreeList = true
void Automat::NoFreeInstances()
{
}
```

```

// Metoda koja inicijalizuje funkcije prelaza i definise vremenske kontrole
// koje ce automati ove klase koristiti u svom radu. Ova metoda se poziva
// implicitno u okviru procesa dodavanja instanci automata u sistem automata.
// Sve funkcije prelaza se posebno deklarise i definisu.
void Automat::Initialize()
{
    // Ovde ide serija inicijalizacija:
    // InitEventProc(uint8 state, uint16 event, PROC_FUN_PTR fun);
    // InitUnexpectedEventProc(uint8 state, PROC_FUN_PTR fun);
    // InitTimerBlock(uint16 timerId, uint32 timerCount, uint16 signalId);

    InitEventProc(IDLE, IDLE_START, ( PROC_FUN_PTR )
        &Automat::Automat_IDLE_START );
    InitEventProc(IDLE, IDLE_MSG, ( PROC_FUN_PTR )
        &Automat::Automat_IDLE_MSG );

    InitEventProc(MESSAGE, MSG_MSG, ( PROC_FUN_PTR )
        &Automat::Automat_MSG_MSG );
    InitEventProc(MESSAGE, MSG_STOP, ( PROC_FUN_PTR )
        &Automat::Automat_MSG_STOP );

    InitUnexpectedEventProc( IDLE, ( PROC_FUN_PTR )
        &Automat::Automat_UNEXPECTED_IDLE );
    InitUnexpectedEventProc( MESSAGE, ( PROC_FUN_PTR )
        &Automat::Automat_UNEXPECTED_MSG );
}

// Funkcije prelaza za stanje IDLE
void Automat::Automat_IDLE_START()
{
    msgNumber = 1;
    idToMsg = GetObjectId()+1;

    // Automati salju poruke u krug
    if( idToMsg == 3 )
        idToMsg = 0;

    // Postavljamo automat u stanje MSG i
    // saljemo poruku sledecem automatu
    PrepareNewMessage( 0x00, IDLE_MSG );

    char text[] = "OVO JE PRVA PORUKA";
    AddParam( PARAM_TEXT, strlen(text), (unsigned char *)text);
    AddParamDWord( COUNT, msgNumber );

    SetMsgToAutomate( FSM_TYPE_AUTOMAT );
    SetMsgToGroup( INVALID_08 );
    SetMsgObjectNumberTo( idToMsg );
    SendMessage( MBX_AUTOMAT_ID );
    SetState( MESSAGE );
}

void Automat::Automat_IDLE_MSG()
{
    idToMsg = GetObjectId()+1;

    // Automati salju poruke u krug
    if( idToMsg == 3 )
        idToMsg = 0;

    // Preuzimamo parametre iz poruke

```



```

unsigned char *tmp;
tmp = GetParam( PARAM_TEXT );
assert( tmp );
memcpy( text, tmp+2, *(tmp+1) );
memset( text+*(tmp+1), 0x00, 1 ); // pravimo string

GetParamDWord( COUNT, msgNumber );

// Automati salju poruke u krug
uint32 idFromMsg = GetObjectId()-1;
if( idFromMsg == -1 )
    idFromMsg = 2;

printf("Primljeni tekst: %s\n od automata:%u \n",text,idFromMsg );

// Ako je broj poruka manji od zadatog
// nastavi sa brojanjem u suprotnom kraj
if( msgNumber < MAX_MSG_NUM )
{
    msgNumber++;
    if( msgNumber < NUM_AUTOMATA )
    {
        // Postavljamo automat u stanje MSG i
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, IDLE_MSG );

        char text[] = "OVO JE DRUGA PORUKA";
        AddParam( PARAM_TEXT, strlen(text), (unsigned char *)text);
        AddParamDWord( COUNT, msgNumber );

        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        SendMessage( MBX_AUTOMAT_ID );
    }
    else
    {
        // Postavljamo automat u stanje MSG i
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, MSG_MSG );
        AddParamDWord( COUNT, msgNumber );
        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        SendMessage( MBX_AUTOMAT_ID );
    }
    SetState( MESSAGE );
}
else
{
    // Postavljamo automat u stanje IDLE i
    // saljemo poruku sledecem automatu
    PrepareNewMessage( 0x00, MSG_STOP );
    AddParamDWord( COUNT, NUM_AUTOMATA );
    SetMsgToAutomate( FSM_TYPE_AUTOMAT );
    SetMsgToGroup( INVALID_08 );
    SetMsgObjectNumberTo( idToMsg );
    SendMessage( MBX_AUTOMAT_ID );

    SetState( IDLE );
}
}

```

```

void Automat::Automat_MSG_MSG()
{
    GetParamDWord( COUNT, msgNumber );
    msgNumber++;
    if( msgNumber < MAX_MSG_NUM )
    {
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, MSG_MSG );
        AddParamDWord( COUNT, msgNumber );
        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        SendMessage( MBX_AUTOMAT_ID );
    }
    else
    {
        printf("Stop automat:%u poruka:%u\n", GetObjectId(), msgNumber );

        // Postavljamo automat u stanje IDLE i
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, MSG_STOP );
        AddParamDWord( COUNT, NUM_AUTOMATA-1 );
        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        SendMessage( MBX_AUTOMAT_ID );
        SetState( IDLE );
    }
}

void Automat::Automat_MSG_STOP()
{
    printf("Stop automat: %u\n", GetObjectId() );

    GetParamDWord( COUNT, msgNumber );
    msgNumber--;
    if( msgNumber > 0 )
    {
        // Postavljamo automat u stanje IDLE i
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, MSG_STOP );
        AddParamDWord( COUNT, msgNumber );
        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        SendMessage( MBX_AUTOMAT_ID );
    }
    SetState( IDLE );
}

void Automat::Automat_UNEXPECTED_IDLE()
{
    printf("Neocekivana poruka u stanju IDLE \n");
}

void Automat::Automat_UNEXPECTED_MSG()
{
    printf("Neocekivana poruka u stanju SEND \n");
}

void Automat::StartDemo()
{
    uint8 *msg = GetBuffer( MSG_HEADER_LENGTH );

```

```

SetMsgFromAutomate( FSM_TYPE_AUTOMAT, msg );
SetMsgFromGroup( INVALID_08, msg );
SetMsgObjectNumberFrom( 0, msg );

SetMsgToAutomate( FSM_TYPE_AUTOMAT, msg );
SetMsgToGroup( INVALID_08, msg );
SetMsgObjectNumberTo( 0, msg );

SetMsgInfoCoding( 0, msg ); // 0 = StandardMessage
SetMsgCode( IDLE_START, msg );
SetMsgInfoLength( 0, msg );
SendMessage( MBX_AUTOMAT_ID, msg );
}

```

Datoteka Konstante.h

```

// FSM
#define FSM_TYPE_AUTOMAT 0

// MBX
#define MBX_AUTOMAT_ID 0

#define MAX_MSG_NUM 10
#define NUM_AUTOMATA 3
#define COUNT 1
#define PARAM_TEXT 2

enum StanjaAutomata
{
    IDLE = 0,
    MESSAGE,
};

enum Poruke
{
    IDLE_START = 0,
    IDLE_MSG,
    MSG_MSG,
    MSG_STOP
};

```

Datoteka Main.cpp

```

#include "conio.h"
#include "Kernel/fsmsystem.h"
#include "Kernel/LogFile.h"
#include "Automat.h"

// Sistem automata sastoji se od jdnog tipa automata koji
// razmenjuje poruke preko jednog postanskog sanduceta
FSMSysSystem sistemAutomata(1,1);

// Pravimo tri instance klase Automat
Automat automat_1, automat_2, automat_3;

```

```

DWORD WINAPI ThreadFunction(void* dummy){
    uint32 buffersCount[3] = { 5, 3, 2 };
    uint32 buffersLength[3] = { 128, 256, 512 };
    uint8  buffClassNo      = 3;

    // Inicijalizacija sistema automata
    printf("Inicijalizujem FSMSystem \n");
    sistemAutomata.Add( &automat_1, FSM_TYPE_AUTOMAT, 3, false );
    sistemAutomata.Add( &automat_2, FSM_TYPE_AUTOMAT );
    sistemAutomata.Add( &automat_3, FSM_TYPE_AUTOMAT );

    sistemAutomata.InitKernel( buffClassNo, buffersCount, buffersLength, 1);

    LogFile lf("log.log", "log.ini");
    LogAutomateNew::SetLogInterface(&lf);

    // Pokrecemo sistem automata
    printf( "Startujem FSMSystem\n" );
    try {
        sistemAutomata.Start();
    }
    catch(...) {
        OutputDebugString( "Exception - prekid rada sistema automata\n" );
        return 0;
    }
    OutputDebugString( "Kraj rada sistema automata\n" );
    return 0;
}

void main( int argc, char* argv[] ){
    DWORD  threadID;
    bool end = false;
    char ret;

    // Pokrecemo obradu poruka sistema automata u posebnoj programskoj niti
    HANDLE hTemp = CreateThread( NULL,0,ThreadFunction,NULL,0,&threadID );
    Sleep( 100 );

    // Program radi dok se sa tastature ne unese karakter Q ili q
    while( !end ){
        if( _kbhit() ) {
            ret = _getch();
            switch( ret ) {
                case 'Q':
                case 'q':
                    sistemAutomata.StopSystem();
                    end = true;
                    Sleep( 100 );
                    break;
                case 'S':
                case 's':
                    automat_1.StartDemo();
                    break;
                default:
                    break;
            }
        }
    }
    CloseHandle( hTemp );
    printf( "Kraj rada \n" );
}

```

10. Primer realizacije NetFSM

U ovom dodatku biće prikazan primer implementacije NetFSM automata i sistema automata sa podrškom za TCP/IP protokol. Ovaj primer je po funkciji sličan predhodnom ali je realizovan preko dva sistema automata (program se instancira dva puta). Automati pripadaju istoj klasi i njihov zadatak je nakon prve poruke da razmenjuju poruke dok se ne dostigne broj zadati broj poruka. Programski kod se može iskoristiti za oba sistema automata ali se mora voditi računa o IP adresama i portovima koji će biti zauzeti. Ovo zavisi da li se programi pokreću na istom računaru ili na dva računara preko mreže.

Datoteka NetAutomat.h

```
#ifndef __NET_AUTOMAT__
#define __NET_AUTOMAT__

#include <stdio.h>
#include "stdlib.h"
#include "kernel\NetFSM.h"
#include "kernel\errorObject.h"
#include "Konstante.h"

class NetAutomat: public NetFSM {
private:
    // NetFSM
    uint16 convertNetToFSMMessage();
    void convertFSMToNetMessage();
    uint8 getProtocolInfoCoding();
    // FSM
    StandardMessage StandardMsgCoding;
    MessageInterface *GetMessageInterface( uint32 id );
    void SetDefaultHeader( uint8 infoCoding );
    uint8 GetMbxId();
    uint8 GetAutomate();
    void SetDefaultFSMData();
    void NoFreeInstances();

    uint8 text[20];
    uint32 msgNumber;
    uint32 idToMsg;

    // Fuunkcije prelaza stanje IDLE
    void NetAutomat_IDLE_START();
    void NetAutomat_IDLE_MSG();
    // Stanje MSG
    void NetAutomat_MSG_MSG();
    void NetAutomat_MSG_STOP();
    // Neocekivane poruke u stanjima IDLE i MSG
    void NetAutomat_UNEXPECTED_IDLE();
    void NetAutomat_UNEXPECTED_MSG();

public:
    NetAutomat();
    ~NetAutomat(){};
    void Initialize();
    void StartDemo();
};
#endif
```

Datoteka NetAutomat.cpp

```

#include "kernel/LogFile.h"
#include "NetAutomat.h"

NetAutomat::NetAutomat() : NetFSM(
    0, //uint16 numOfTimers = DEFAULT_TIMER_NO,
    2, //uint16 numOfState = DEFAULT_STATE_NO,
    3 )//uint16 maxNumOfProceduresPerState = DEFAULT_PROCEDURE_NO_PER_STATE
{
    SetDefaultFSMData();
}

// Ova metoda sadrzi informacije o tome koje poruke automat može da obradi
// Za ovakvu definiciju u okviru klase potrebno je definisati dodatnu
// promenljivu: StandardMessage standardMsgCoding;
MessageInterface *NetAutomat::GetMessageInterface( uint32 id ){
    switch( id ) {
        case 0x00:
            return &StandardMsgCoding;
    }
    throw TErrorObject( __LINE__, __FILE__, 0x01010400);
}

// Ova metoda popunjava zaglavlje poruke koju automati međusobno razmenjuju
void NetAutomat::SetDefaultHeader( uint8 infoCoding ){
    SetMsgInfoCoding( infoCoding );
    SetMessageFromData();
}

// Metoda kojom se definiše broj postanskog sanduceta iz kog će automati ove
// klase primiti poruke koje su pristigle u sistem automata.
uint8 NetAutomat::GetMbxId(){
    return MBX_AUTOMAT_ID;
}

// Metoda vraća broj koji identifikuje tip automata kojem pripada ovaj automat
uint8 NetAutomat::GetAutomate(){
    return FSM_TYPE_AUTOMAT;
}

// Metoda u kojoj se izvršava inicijalizacija atributa instanci ove klase.
void NetAutomat::SetDefaultFSMData(){
    msgNumber = 0;
    idToMsg = INVALID_32;
}

// Metoda koja će se pozvati ako nema više slobodnih automata da obrade
// pristiglu poruku. Ova metoda se koristi ako su automati ove klase prethodno
// dodati u sistem automata sa parametrom useFreeList = true
void NetAutomat::NoFreeInstances(){}

// Metoda koja inicijalizuje funkcije prelaza i definiše vremenske kontrole
// koje će automati ove klase koristiti u svom radu. Ova metoda se poziva
// implicitno u okviru procesa dodavanja instanci automata u sistem automata.
// Sve funkcije prelaza se posebno deklarise i definišu.
void NetAutomat::Initialize(){
    // Ovde ide serija inicijalizacija:
    // InitEventProc(uint8 state, uint16 event, PROC_FUN_PTR fun);
    // InitUnexpectedEventProc(uint8 state, PROC_FUN_PTR fun);
    // InitTimerBlock(uint16 timerId, uint32 timerCount, uint16 signalId);

    InitEventProc(IDLE, IDLE_START, ( PROC_FUN_PTR )

```

```

        &NetAutomat::NetAutomat_IDLE_START );
    InitEventProc( IDLE, IDLE_MSG, ( PROC_FUN_PTR )
        &NetAutomat::NetAutomat_IDLE_MSG );

    InitEventProc( MESSAGE, MSG_MSG, ( PROC_FUN_PTR )
        &NetAutomat::NetAutomat_MSG_MSG );
    InitEventProc( MESSAGE, MSG_STOP, ( PROC_FUN_PTR )
        &NetAutomat::NetAutomat_MSG_STOP );

    InitUnexpectedEventProc( IDLE, ( PROC_FUN_PTR )
        &NetAutomat::NetAutomat_UNEXPECTED_IDLE );
    InitUnexpectedEventProc( MESSAGE, ( PROC_FUN_PTR )
        &NetAutomat::NetAutomat_UNEXPECTED_MSG );
}

// Funkcije prelaza za stanje IDLE
void NetAutomat::NetAutomat_IDLE_START(){
    msgNumber = 1;
    idToMsg = 0;

    // Postavljamo automat u stanje MSG i
    // saljemo poruku sledecem automatu
    PrepareNewMessage( 0x00, IDLE_MSG );

    char text[] = "OVO JE PRVA PORUKA";
    AddParam( PARAM_TEXT, strlen(text), (unsigned char *)text);
    AddParamDWord( COUNT, msgNumber );

    SetMsgToAutomate( FSM_TYPE_AUTOMAT );
    SetMsgToGroup( INVALID_08 );
    SetMsgObjectNumberTo( idToMsg );
    sendToTCP();
    SetState( MESSAGE );
}

void NetAutomat::NetAutomat_IDLE_MSG(){
    idToMsg = 0;

    // Preuzimamo parametre iz poruke
    unsigned char *tmp;
    tmp = GetParam( PARAM_TEXT );
    assert( tmp );
    memcpy( text, tmp+2, *(tmp+1) );
    memset( text+*(tmp+1), 0x00, 1 ); // pravimo string

    GetParamDWord( COUNT, msgNumber );
    printf("Primljeni tekst: %s\n", text);

    // Ako je broj poruka manji od zadatog
    // nastavi sa brojanjem u suprotnom kraj
    if( msgNumber < MAX_MSG_NUM ){
        msgNumber++;

        // Postavljamo automat u stanje MSG i
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, MSG_MSG );
        AddParamDWord( COUNT, msgNumber );
        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        sendToTCP();

        SetState( MESSAGE );
    }
}

```

```

    }
    else {
        printf("Stop automat: %u, poruka %u\n", GetObjectId(), msgNumber );

        // Postavljamo automat u stanje IDLE i
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, MSG_STOP );
        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        sendToTCP();
        SetState( IDLE );
    }
}

void NetAutomat::NetAutomat_MSG_MSG(){
    GetParamDWord( COUNT, msgNumber );
    msgNumber++;
    if( msgNumber < MAX_MSG_NUM ){
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, MSG_MSG );
        AddParamDWord( COUNT, msgNumber );
        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        sendToTCP();
    }
    else {
        printf("Stop automat: %u\n", GetObjectId() );

        // Postavljamo automat u stanje IDLE i
        // saljemo poruku sledecem automatu
        PrepareNewMessage( 0x00, MSG_STOP );
        SetMsgToAutomate( FSM_TYPE_AUTOMAT );
        SetMsgToGroup( INVALID_08 );
        SetMsgObjectNumberTo( idToMsg );
        sendToTCP();
        SetState( IDLE );
    }
}

void NetAutomat::NetAutomat_MSG_STOP(){
    printf("Stop automat: %u\n", GetObjectId() );
    SetState( IDLE );
}

void NetAutomat::NetAutomat_UNEXPECTED_IDLE(){
    printf("Neocekivana poruka u stanju IDLE \n");
}

void NetAutomat::NetAutomat_UNEXPECTED_MSG(){
    printf("Neocekivana poruka u stanju SEND \n");
}

void NetAutomat::StartDemo(){
    uint8 *msg = GetBuffer( MSG_HEADER_LENGTH );
    SetMsgFromAutomate( FSM_TYPE_AUTOMAT, msg );
    SetMsgFromGroup( INVALID_08, msg );
    SetMsgObjectNumberFrom( 0, msg );

    SetMsgToAutomate( FSM_TYPE_AUTOMAT, msg );
    SetMsgToGroup( INVALID_08, msg );
    SetMsgObjectNumberTo( 0, msg );
}

```



```

        SetMsgInfoCoding( 0, msg ); // 0 = StandardMessage
        SetMsgCode( IDLE_START, msg );
        SetMsgInfoLength( 0, msg );
        SendMessage( MBX_AUTOMAT_ID, msg );
    }

uint16 NetAutomat::convertNetToFSMMessage(){
    // Ovde je potrebno poslati samo podatke posto se nova
    // poruka salje samom sebi
    int length = receivedMessageLength-MSG_HEADER_LENGTH;
    memcpy(fsmMessageR, protocolMessageR+MSG_HEADER_LENGTH, length);
    fsmMessageRLength = length; //obavezno postaviti jer to koristi
    workWhenReceive()

    // Rotiramo bajtove
    uint16 msgCode = GetUint16((uint8*)(protocolMessageR+MSG_CODE));

    switch( msgCode ){
        case IDLE_START:
            msgCode = IDLE_START;
            break;
        case IDLE_MSG:
            msgCode = IDLE_MSG;
            break;
        case MSG_MSG:
            msgCode = MSG_MSG;
            break;
        case MSG_STOP:
            msgCode = MSG_STOP;
            break;
        default:
            msgCode = 0xffff;
    }
    return msgCode;
}

void NetAutomat::convertFSMToNetMessage(){
    // Ovde se salje cela poruka
    memcpy( protocolMessageS, fsmMessageS, fsmMessageSLength );
    sendMsgLength = fsmMessageSLength;
}

uint8 NetAutomat::getProtocolInfoCoding(){
    // Standard msg info coding
    return 0;
}

```

Datoteka Konstante.h

```

// FSM
#define FSM_TYPE_AUTOMAT    0

// MBX
#define MBX_AUTOMAT_ID     0

#define MAX_MSG_NUM        10
#define NUM_AUTOMATA       3
#define COUNT              1
#define PARAM_TEXT         2
#define IP_ADDRES          "192.168.0.57"

```

```

#define PORT_1          7000
#define PORT_2          8000

enum StanjaAutomata
{
    IDLE = 0,
    MESSAGE,
};

enum Poruke
{
    IDLE_START = 0,
    IDLE_MSG,
    MSG_MSG,
    MSG_STOP
};

```

Datoteka Main.cpp

```

#include "conio.h"
#include "Kernel/fsmsystem.h"
#include "Kernel/LogFile.h"
#include "NetAutomat.h"

// Ovom definicijom modifikujemo program ako treba da se naprave
// dva programa koji ce biti pokrenuti na istom racunaru
#define AUTOMAT1

// Sistem automata sastoji se od jednog tipa automata koji
// razmenjuje poruke preko jednog postanskog sanduceta
FSMSystemWithTCP sistemAutomata(1,1);

// Pravimo instancu klase NetAutomat
NetAutomat automat_1;

DWORD WINAPI ThreadFunction(void* dummy){
    uint32 buffersCount[3] = { 5, 3, 2 };
    uint32 buffersLength[3] = { 128, 256, 512 };
    uint8 buffClassNo      = 3;

    // Inicijalizacija sistema automata
    printf("Inicijalizujem FSMSystemWithTCP \n");
    sistemAutomata.Add( &automat_1, FSM_TYPE_AUTOMAT, 1, true );
    sistemAutomata.InitKernel( buffClassNo, buffersCount, buffersLength, 1);
    LogFile lf("log.log", "log.ini");
    LogAutomateNew::SetLogInterface(&lf);

    // Kako je svejedno ko ce pokrenuti slanje poruka u oba automata
    // je uveden server stim sto je za program 1 server na portu PORT_1
    // a za drugi program na portu PORT_2. Sve je jedno koji ce automat prvi
    // pokrenuti TCP komunikaciju sa porukom establishConection().
#ifdef AUTOMAT1
    printf("Startujemo server...na portu:%u\n", PORT_1 );
    sistemAutomata.InitTCPServer( PORT_1, FSM_TYPE_AUTOMAT );
#else
    printf("Startujemo server...na portu:%u\n", PORT_2 );
    sistemAutomata.InitTCPServer( PORT_2, FSM_TYPE_AUTOMAT );
#endif

    // Pokrecemo sistem automata
}

```

```

    printf( "Startujem FSMSystem\n" );
    try {
        sistemAutomata.Start();
    }
    catch(...) {
        OutputDebugString( "Exception - prekid rada sistema automata\n" );
        return 0;
    }
    OutputDebugString( "Kraj rada sistema automata\n" );
    return 0;
}

void main( int argc, char* argv[] ){
    DWORD  threadID;
    bool end = false;
    char ret;

    // Pokrecemo obradu poruka sistema automata u posebnoj programskoj niti
    HANDLE hTemp = CreateThread( NULL,0,ThreadFunction,NULL,0,&threadID );
    Sleep( 100 );

    // Program radi dok se sa tastature ne unese karakter Q ili q
    while( !end ) {
        if( _kbhit() ) {
            ret = _getch();
            switch( ret ) {
                case 'Q':
                case 'q':
                    sistemAutomata.StopSystem();
                    end = true;
                    Sleep( 100 );
                    break;
                case 'S':
                case 's':
                    automat_1.StartDemo();
                    break;
                case 'E':
                case 'e':
                    // Pritiskom na e u bilo kom automatu pokrece se uspostavljanje veze
                    // sa serverom drugog automata kako bi se uspostavila veza za razmenu podataka
#ifdef AUTOMAT1
                    automat_1.setPort( PORT_2 );
                    automat_1.setIP( IP_ADDRES );
                    printf("establishConection...na portu:%u", PORT_2 );
                    automat_1.establishConnection();
#else
                    automat_1.setPort( PORT_1 );
                    automat_1.setIP( IP_ADDRES );
                    printf("establishConection...na portu:%u", PORT_1 );
                    automat_1.establishConnection();
#endif

                default:
                    break;
            }
        }
    }
    CloseHandle( hTemp );
    printf( "Kraj rada \n" );
}

```

Skraćenice

API – Application Programing Interface

FSM – Finite State Machine